



普通高等教育“十一五”国家级规划教材

卓越工程师培养计划系列教材

国家级优秀教学团队教学成果

计算机组成原理 (第3版)

| 纪禄平 刘 辉 罗克露 等编

| 俸远祯 审



电子工业出版社

PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

<http://www.phei.com.cn>

普通高等教育“十一五”国家级规划教材

计算机组成原理

(第3版)

纪禄平 刘 辉 罗克露
俸志刚 张 建 杜向辉 编
俸远祯 审

电子工业出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书为普通高等教育“十一五”国家级规划教材。

本书以当前主流微型计算机技术为背景,以建立系统级的整机概念为目的,深入介绍计算机各功能子系统的逻辑组成和工作机制。全书共7章。第1章概述计算机的基本概念、发展历程和系统的硬件、软件组织及性能指标;第2章介绍数据信息的表示、运算和校验方法;第3章介绍CPU子系统的工作原理和多核等前沿的性能提升技术;第4章介绍存储子系统的存储原理、主存设计和性能改进措施;第5章介绍I/O子系统,包括接口、总线以及中断、DMA和通道等I/O传输控制方式;第6章介绍显示器等常见输入/输出设备的工作原理;第7章以一个硬件系统模型的设计作为全书的总结。

本书可作为高等院校计算机及相关专业“计算机组成原理”及相关课程的教材,也可作为从事计算机专业的工程技术人员的参考书。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究。

图书在版编目(CIP)数据

计算机组成原理/纪禄平等编. —3版. —北京:电子工业出版社,2014.9

ISBN 978-7-121-23471-2

I. ① 计… II. ① 纪… III. ① 计算机组成原理—高等学校—教材 IV. ① TP301

中国版本图书馆CIP数据核字(2014)第122802号

策划编辑:章海涛

责任编辑:章海涛 特约编辑:何 雄

印 刷:

装 订:

出版发行:电子工业出版社

北京市海淀区万寿路173信箱 邮编 100036

开 本:787×1092 1/16 印张:26.5 字数:740千字

版 次:2004年9月第1版

2014年9月第3版

印 次:2014年9月第1次印刷

定 价:45.00元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888。

质量投诉请发邮件至 zltz@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线:(010) 88258888。

第 3 版前言

本书第 1 版（罗克露等主编，2004 年）是在《计算机组成原理》（俸远祯等主编，“六五”国家级规划教材，1985 年）、《计算机组成原理（修订本）》（俸远祯等主编，“八五”国家级规划教材、教育部科技进步奖教材类三等奖，1996 年）和《计算机组成原理与汇编语言程序设计》（俸远祯等主编，“九五”国家级规划教材，1997 年）这三本国家级规划教材的基础上进行编写的。

本书在《计算机组成原理（第 2 版）》（罗克露等主编，普通高等教育“十一五”国家级规划教材，2010 年）基础上进行修订，以当前主流微机技术为背景，以建立系统级整机概念为目的，深入介绍计算机各功能子系统的逻辑组成和工作机制。

全书共 7 章，按照先建立 CPU 整机概念，再发展为主机，最后形成硬件系统的教学思路来安排各章节的内容。

第 1 章概述计算机的概念、发展历程和系统的硬件、软件组织，强调信息的数字化表示和存储程序方式这两个重要概念。第 2 章介绍数据信息的表示、运算和校验方法。第 3 章介绍 CPU 子系统及多核等前沿的性能提升技术，通过一个 CPU 简化模型，深入分析指令的执行过程，同时阐明组合逻辑控制和微程序控制的设计方法。第 4 章介绍存储子系统，讨论存储原理和主存储器的设计，以及提高存储系统性能的一些经典措施。第 5 章介绍输入/输出子系统，包括接口和总线等，并强调中断、DMA 和通道等 I/O 传输的控制机制。第 6 章介绍常见输入/输出设备的工作原理，包括显示器和打印机等。作为全书的总结，第 7 章给出一个计算机硬件系统模型，并描述它的组织结构和典型的 I/O 操作。

本书第 1 章、第 2 章由刘辉编写，第 3 章由罗克露编写，第 4 章由纪禄平编写，第 5 章由张建编写，第 6 章由杜向辉编写，第 7 章由俸志刚编写。全书由纪禄平统稿，电子科技大学俸远祯教授主审。作为相关课程国家级规划教材的前辈，俸远祯教授和罗克露教授对本书的编写给予了大力支持和悉心指导，在此谨向两位教授致敬！

本书在编写过程中不仅得到了电子科技大学教务处和计算机学院相关领导的支持与鼓励，还得到了电子工业出版社的积极协助，在此谨向他们致以诚挚谢意！

本书还有配套的辅导书——《计算机组成原理课程设计》，其中包含与本书相关的课程设计、学习指导以及习题解答和电子竞技等内容。

由于编者的水平有限，书中难免会有错误和不当之处，恳请读者批评指正，我们不胜感激。如有问题请直接与编者邮件联系：jl0813@163.com。

本书为任课教师提供配套的教学资源（包含电子教案），需要者可[登录华信教育资源网](http://www.hxedu.com.cn)站（<http://www.hxedu.com.cn>），注册之后进行[免费下载](#)，或发邮件到 unicode@phei.com.cn 进行[咨询](#)（注明所在的学校及院系）。

作 者

目 录

| | |
|----------------------|----|
| 第 1 章 概论 | 1 |
| 1.1 计算机的基本概念 | 1 |
| 1.1.1 信息的数字化表示 | 2 |
| 1.1.2 存储程序工作方式 | 5 |
| 1.1.3 计算机的分类 | 6 |
| 1.2 计算机的诞生和发展 | 8 |
| 1.2.1 冯·诺依曼体系 | 8 |
| 1.2.2 计算机发展历程 | 9 |
| 1.2.3 未来的发展趋势 | 12 |
| 1.3 计算机系统的组织 | 13 |
| 1.3.1 硬件系统 | 13 |
| 1.3.2 软件系统 | 18 |
| 1.3.3 硬件、软件系统层次结构 | 20 |
| 1.3.4 硬件、软件功能划分与逻辑等价 | 23 |
| 1.4 计算机的特点与性能指标 | 24 |
| 1.4.1 计算机的特点 | 24 |
| 1.4.2 计算机的主要性能指标 | 24 |
| 习题 1 | 28 |
| 第 2 章 数据的表示、运算与校验 | 29 |
| 2.1 数值型数据的表示 | 29 |
| 2.1.1 进位计数制 | 29 |
| 2.1.2 带符号数的表示 | 35 |
| 2.1.3 定点数与浮点数 | 40 |
| 2.2 字符型数据的表示 | 46 |
| 2.3 运算方法 | 48 |
| 2.3.1 定点加减运算 | 48 |
| 2.3.2 溢出的判断与移位 | 50 |
| 2.3.3 定点乘法运算 | 53 |
| 2.3.4 定点除法运算 | 60 |
| 2.3.5 浮点四则运算 | 65 |
| 2.4 常用的数据校验方法 | 67 |
| 2.4.1 奇偶校验 | 68 |
| 2.4.2 海明校验 | 69 |
| 2.4.3 循环冗余校验 | 71 |

| | |
|-------------------------------|-----------|
| 习题 2 | 73 |
| 第 3 章 CPU 子系统 | 75 |
| 3.1 概述 | 75 |
| 3.1.1 CPU 的基本组成 | 75 |
| 3.1.2 CPU 的工作原理 | 81 |
| 3.1.3 CPU 的指令集类型 | 82 |
| 3.1.4 CPU 的发展历程 | 83 |
| 3.2 指令系统 | 86 |
| 3.2.1 指令格式 | 86 |
| 3.2.2 寻址方式 | 92 |
| 3.2.3 指令的功能和类型 | 107 |
| 3.3 CPU 的基本模型 | 115 |
| 3.3.1 CPU 设计步骤 | 116 |
| 3.3.2 模型机的指令系统 | 116 |
| 3.3.3 模型机的组成与数据通路 | 120 |
| 3.4 运算部件 | 124 |
| 3.4.1 加法单元 | 124 |
| 3.4.2 加法器与进位逻辑 | 125 |
| 3.4.3 算术逻辑运算部件 | 128 |
| 3.4.4 运算器的组织 | 132 |
| 3.5 组合逻辑控制方式 | 134 |
| 3.5.1 组合逻辑控制器时序系统 | 135 |
| 3.5.2 指令流程与操作时间表 | 136 |
| 3.5.3 微命令的综合与产生 | 146 |
| 3.6 微程序控制方式 | 148 |
| 3.6.1 微程序控制的基本原理 | 148 |
| 3.6.2 微指令编码与微地址形成 | 150 |
| 3.6.3 模型机微指令格式 | 153 |
| 3.6.4 模型机的微程序设计 | 155 |
| 3.7 CPU 性能的提升技术 | 162 |
| 3.7.1 流水线技术 | 162 |
| 3.7.2 SMT 与超线程 | 167 |
| 3.7.3 多核技术 | 168 |
| 3.8 经典处理器介绍 | 171 |
| 3.8.1 Intel 8086/8088 | 172 |
| 3.8.2 Intel 80386/80486 | 175 |
| 3.8.3 Pentium 微处理器 | 178 |
| 3.8.4 Alpha 微处理器 | 180 |

| | | |
|-------|-------------|-----|
| 3.8.5 | CRAY-1 | 181 |
| 3.8.6 | Transputer | 183 |
| 习题 3 | | 184 |
| 第 4 章 | 存储子系统 | 186 |
| 4.1 | 概述 | 186 |
| 4.1.1 | 存储系统的层次结构 | 186 |
| 4.1.2 | 物理存储器与虚拟存储器 | 190 |
| 4.1.3 | 存储器的分类 | 190 |
| 4.1.4 | 存储器的技术指标 | 193 |
| 4.2 | 半导体存储原理 | 194 |
| 4.2.1 | 双极型存储单元 | 194 |
| 4.2.2 | 静态 MOS 存储单元 | 197 |
| 4.2.3 | 动态 MOS 存储单元 | 201 |
| 4.2.4 | 半导体只读存储器 | 206 |
| 4.3 | 主存储器的组织 | 210 |
| 4.3.1 | 主存的设计原则 | 210 |
| 4.3.2 | 主存的逻辑设计 | 212 |
| 4.3.3 | 主存的外部连接方式 | 214 |
| 4.3.4 | 常见的主存芯片技术 | 217 |
| 4.3.5 | 存储器的刷新与校验 | 220 |
| 4.4 | 磁表面存储原理 | 223 |
| 4.4.1 | 存储介质与磁头 | 223 |
| 4.4.2 | 读写原理 | 224 |
| 4.4.3 | 磁记录的编码方式 | 226 |
| 4.5 | 磁盘存储器及其接口 | 230 |
| 4.5.1 | 软盘存储器 | 231 |
| 4.5.2 | 硬盘存储器 | 234 |
| 4.5.3 | 技术指标与数据校验 | 243 |
| 4.5.4 | 磁盘适配器 | 245 |
| 4.5 | 光学存储及器件 | 248 |
| 4.5.1 | 光存储原理 | 248 |
| 4.5.2 | 光盘存储器 | 250 |
| 4.5.3 | 光盘驱动器及其发展方向 | 251 |
| 4.6 | 存储系统性能的改进措施 | 254 |
| 4.6.1 | 高速缓冲存储器 | 255 |
| 4.6.2 | 虚拟存储器 | 261 |
| 4.6.3 | 双端口存储器 | 266 |
| 4.6.4 | 并行存储器 | 266 |

| | | |
|--------------|--------------------------|------------|
| 4.6.5 | 联想存储器 | 270 |
| 习题 4 | | 272 |
| 第 5 章 | 总线与输入/输出子系统 | 274 |
| 5.1 | 概述 | 274 |
| 5.1.1 | 总线简介 | 275 |
| 5.1.2 | 接口的功能与类型 | 276 |
| 5.1.3 | 输入/输出控制方式 | 279 |
| 5.2 | 总线 | 281 |
| 5.2.1 | 总线的特性与分类 | 282 |
| 5.2.2 | 总线的标准 | 284 |
| 5.2.3 | 总线的设计要素 | 285 |
| 5.2.4 | PCI 总线介绍 | 292 |
| 5.3 | 直接程序传输方式与接口 | 295 |
| 5.4 | 中断方式与接口 | 296 |
| 5.4.1 | 中断的相关概念 | 296 |
| 5.4.2 | 中断请求 | 301 |
| 5.4.3 | 中断判优 | 302 |
| 5.4.4 | 中断响应 | 306 |
| 5.4.5 | 中断处理 | 309 |
| 5.4.6 | 中断接口组成模型 | 312 |
| 5.4.7 | 典型中断接口举例 | 315 |
| 5.5 | DMA 方式与接口 | 318 |
| 5.5.1 | DMA 方式基本概念 | 318 |
| 5.5.2 | DMA 控制器与接口的连接 | 321 |
| 5.5.3 | DMA 控制器的组成 | 325 |
| 5.5.4 | DMA 传输操作过程 | 329 |
| 5.5.5 | 典型 DMA 接口举例 | 330 |
| 5.6 | IOP 和 PPU | 335 |
| 5.6.1 | 通道的系统结构 | 335 |
| 5.6.2 | 通道的类型 | 336 |
| 5.6.3 | 通道的工作原理 | 337 |
| 习题 5 | | 340 |
| 第 6 章 | 输入/输出设备及接口 | 341 |
| 6.1 | 概述 | 341 |
| 6.1.1 | 输入/输出设备的一般功能 | 341 |
| 6.1.2 | 输入/输出设备的类型 | 342 |
| 6.1.3 | 输入/输出设备与主机之间的信息交换 | 345 |
| 6.2 | 键盘及接口 | 346 |

| | | |
|-------|------------------|-----|
| 6.2.1 | 键盘的类型 | 346 |
| 6.2.2 | 硬件扫描键盘 | 349 |
| 6.2.3 | 软件扫描键盘 | 350 |
| 6.3 | 显示设备及接口 | 353 |
| 6.3.1 | 显示器的分类 | 353 |
| 6.3.2 | 显示器的成像原理 | 354 |
| 6.3.3 | CRT 显示器 | 361 |
| 6.3.4 | LCD 显示器 | 367 |
| 6.3.5 | 显示适配器及工作原理 | 374 |
| 6.4 | 打印设备及接口 | 378 |
| 6.4.1 | 概述 | 378 |
| 6.4.2 | 打印机的性能指标 | 380 |
| 6.4.3 | 点阵针式打印机 | 381 |
| 6.4.4 | 喷墨打印机 | 385 |
| 6.4.5 | 激光打印机 | 387 |
| 6.4.6 | 打印机适配器 | 390 |
| 6.4.7 | 3D 打印技术简介 | 391 |
| 6.5 | 其他输入/输出设备 | 392 |
| 6.5.1 | 光学字符识别设备 | 392 |
| 6.5.2 | 图形图像输入设备 | 393 |
| 6.5.3 | 语音识别设备 | 395 |
| 6.5.4 | 条形码与二维码识别仪 | 396 |
| 习题 6 | | 399 |
| 第 7 章 | 计算机硬件系统模型 | 400 |
| 7.1 | 模型机系统及信号互连 | 400 |
| 7.1.1 | 系统组成 | 400 |
| 7.1.2 | 系统总线 | 401 |
| 7.1.3 | 各部件的信号线 | 402 |
| 7.2 | 模型机典型 I/O 操作举例 | 406 |
| 7.2.1 | 直接程序传输方式的 I/O 操作 | 406 |
| 7.2.2 | 中断方式下的 I/O 操作 | 407 |
| 7.2.3 | DMA 方式下的 I/O 操作 | 408 |
| 7.3 | 系统配置举例 | 409 |
| 习题 7 | | 411 |
| 参考文献 | | 412 |

第 1 章 概 论

“计算机组成原理”课程的主要内容是以单机系统为对象，阐述计算机系统的硬件组成，其核心是建立一个计算机系统的整机概念。这里提到的“整机概念”包括两层面，即 CPU 级的整机和硬件系统级的整机概念，且每个层面都涉及硬件的逻辑组成及其工作原理机制。本书将从这两个层面逐步建立前述的整机概念。为此，本章首先阐明三个重要的基本概念：信息的数字化表示、存储程序工作方式和计算机系统的层次结构，并将这些基本概念作为了解计算机的逻辑组成结构和硬件系统工作原理的基本出发点。

1.1 计算机的基本概念

计算机是 20 世纪人类最伟大的发明之一，它是一种能够按照事先存储的程序，代替人类自动地、高速地进行大量数值计算和处理各种信息的现代化智能电子设备。

计算机系统通常由硬件和软件两大部分组成。硬件是指看得见、摸得着且物理存在的设备实体，如运算器、控制器、存储器和输入/输出设备等，如图 1-1 所示；软件则是指不能直接触摸但又确实在逻辑上存在的对象，如程序和文档等。设计计算机硬件系统的基本原则是功能部件的逻辑化，即用逻辑电路构造各种部件，如用基本的门电路、触发器来构造运算器、控制器、存储器等。在硬件基础之上，根据应用需要配置各种软件，如操作系统、编程语言以及各种支撑软件等等。硬件与软件按层次结构组成一个复杂的计算机系统。

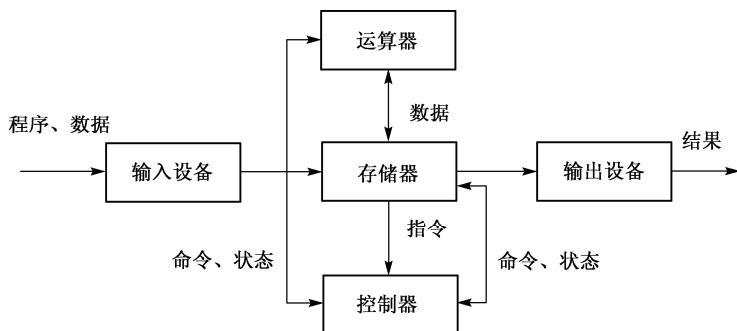


图 1-1 计算机组成示意图

计算机系统是如何工作的呢？不管是做一次复杂的数学计算，还是对大量的数据进行处理，或者对一个过程进行自动控制，用户都应先按照处理的步骤，用编程语言编写程序，然后通过输入设备（如键盘）将程序和需处理的数据送入计算机并存放在存储器中。用户编写的程序称为源程序，是不能被计算机直接执行的。计算机只能执行机器指令，即要求计算机完成某种操作的命令，如执行加法操作的加法指令、执行乘法操作的乘法指令、执行传送操作的传送指令等。因此，计算机在运行程序之前，必须将源程序编译转换为机器指令，并将这些指令按一定顺序存放在存储器的若干个单元中。每个单元对应一个称为地址的固定编号，只要给出确定的地址，就能访问相应的存储单元，对该单元的内容进行读/写操作。

当计算机启动运行后，控制器将某个地址送往存储器，从该地址单元取回一条指令。控制器

根据这条指令的含义，发出相应的操作命令，控制该指令的执行。比如，执行一条加法指令，先要从存储单元或寄存器取出操作数，送入运算器，再将两个操作数相加，并将运算处理的结果送回存储单元或寄存器存放。如果用户要了解处理结果，则计算机可通过输出设备（如显示器、打印机等），将结果显示在屏幕或打印在纸上。图 1-2 给出了计算机的简单工作流程。

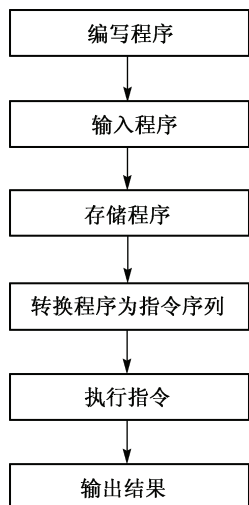


图 1-2 计算机工作流程

从以上描述可看出，计算机作为一个能够自动处理信息的智能工具，它的设计过程必须考虑到很多因素。首先，信息如何表示，才能方便让计算机进行识别和处理。其次，计算机硬件系统应该由哪些部件组成，每部分的相互关系是怎样的，以及如何控制它们协同工作。再次，采用什么样的工作方式，才能使计算机能够自动地对信息进行处理。最后，应该提供怎样的人机交互接口，才能方便操作和使用计算机。

纵观计算机的诞生和发展历程，可以看出上述几个核心问题，自始至终都贯穿于计算机发展的各个阶段中。不同时代对计算机有特定的应用需求，而基于这些需求提出的解决思路，完美地回答了这些问题，从而促进了计算机由弱到强、从低级到高级地向前不断发展。

1.1.1 信息的数字化表示

我们已知道，计算机是通过执行程序（指令序列）来实现对数据的加工处理的。因此，计算机中的信息可以分为两大类：控制信息和数据信息。控制信息用来控制计算机的工作。计算机执行指令时，用指令产生的控制命令（称为微命令）控制有关操作，所以指令序列和微命令序列属于控制信息类。数据信息是计算机加工处理的对象。计算机根据指令要求取出的操作数以及对操作数处理的结果等，都属于数据信息类。数据信息又分为数值型数据和非数值型数据两类。数值型数据有数值大小与正负之分，如 6、-15 等。非数值型数据则无数值大小，也不分正数负数，如字符、文字、图像、声音等人们能够识别的信息，以及条件、状态、命令等用于判定的逻辑信息。那么，在计算机中如何表示这些信息呢？

前面讲过，计算机的主要部件是用逻辑电路，即电子电路构成的，所以，在电子数字计算机中传送与处理的信息都采用数字化表示方法。信息的数字化表示包含了两层含义：① 用数字代码表示各种信息；② 用数字信号表示数字代码。信息表示数字化这一重要概念是我们理解计算机工作原理的一个基本出发点。下面通过几个例子对这两层含义加以说明。

1. 在计算机中用数字代码表示各类信息

数字代码是指一组数字的集合，这里的数字代码通常指二进制数字代码。我们可以根据需要描述的信息（某类控制信息或某类数据信息），用一组约定了含义的数字代码来表示它。

【例 1-1】 用数字代码表示数值型数据。

6 和 7 是两个数值型数据。可以约定，用一位二进制代码表示每个数的符号，如用 0 表示正数，用 1 表示负数；再用 4 位二进制代码表示每个数的大小。这样，代码 00110 表示 6（左边第一个 0 代表正号），代码 10111 表示 -7（左边第一个 1 代表负号）。

当然，也可以用 8 位二进制代码表示一个数的大小。代码位数增多，数的表示范围将扩大。例如，用 4 位二进制代码不能表示 200 这个数，但用 8 位二进制代码则可以表示它。

【例 1-2】用数字代码表示字符。

字符本身没有大小和正负之分，但仍然可以用数字代码来表示它。计算机中常约定用 7 代码表示一个西文字符，如用 1000001 表示字符 A，用 1000010 表示字符 B；或用 7 位代码表示一个控制字符，如用 0001100 表示换页 (FF)，用 0001101 表示回车 (CR)。字符的这种编码称为 ASCII 码，是国际上广泛采用的一种字符表示方法。另外，还可以约定用两组 8 位二进制代码表示一个中文字符，如用 01010110 01010000 表示“中”，用 00111001 01111010 表示“国”等。总之，用数字代码可以表示各种字符，而以字符为基础又可以表示范围广泛的各種文字。

【例 1-3】用数字代码表示图像。

字符的种类总是有限的，因而可以用若干位编码来表示。图像则不然，其变化是无穷无尽的，那么，如何用数字代码来表示这些随机分布的图像信息呢？实际上，一幅图像可以被细分为若干个点，这些点称为像素。也就是说，我们可以用像素的组合来逼近真实的图像。图像划分得越细，像素越多，组成的图像也就越真实。按照信息表示数字化的思想，可以用数字代码表示像素。例如，用一位代码表示一个像素，若像素是亮的，则用代码 1 表示；若像素是暗的，则用代码 0 表示。再将表示一幅图像所有像素的代码按照像素在图像中的位置进行组织，就可以实现用数字代码来表示图像了。

【例 1-4】用数字代码表示声音。

为了对声音信息数字化，首先要将声波转换为电流波，再按一定频率对电流波进行采样，即在长度相同的时间间隔内分别对电流波的幅值进行测量，每次测到的电流幅值都用一个数字量来表示。只要采样频率足够高，所得到的数字信息就能逼真地保持声波信息，还原后真实地再现原来的声音。

【例 1-5】用数字代码表示指令。

指令属于控制信息。通常，一条指令需提供要求计算机做什么操作，以及如何获取操作数等信息。因此，可以用一段数字代码表示操作类型，这段代码称为操作码；用另一段代码表示获取操作数的途径，这段代码称为地址码。将操作码和地址码组合在一起，就形成了机器指令代码。例如，操作码取 4 位，可以约定，0000 表示传送操作，0001 表示加法操作，0010 表示减法操作……地址码取 12 位，用 6 位表示一个操作数的来源，用其余 6 位表示另一个操作数的来源。例如，约定 000000 表示操作数来自 0 号寄存器，000001 表示另一操作数来自 1 号寄存器。这样，16 位代码 0001000000000001 表示一条加法指令，其含义是将 0 号寄存器的内容与 1 号寄存器的内容相加，结果存放在 1 号寄存器中。

【例 1-6】用数字代码表示设备状态。

计算机在工作时往往需要了解外部设备的状态，根据外设状态决定做什么操作。不同的外部设备可能有不同的工作状态，如打印机将字符打印在纸上，而显示器则将字符显示在屏幕上。这些设备的状态可以抽象、归纳为三种：空闲（设备没有工作）、忙（设备正在工作）、完成（设备做完一次操作）。相应地，可以用约定的数字代码表示这三种状态，如用 00 表示空闲，01 表示忙，10 表示完成。

2. 在物理机制上用数字信号表示数字代码

为什么能用数字代码来表示各种信息呢？这就涉及计算机的物理机制。计算机是一种复杂的电子线路，传送和处理的实际对象是电信号。电信号又分为模拟信号和数字信号两种。

模拟信号是一种随时间连续变化的电信号，如电流信号、电压信号等。我们可以用电流或电压的幅值来模拟数值或物理量的大小，如模拟温度的高低、压力的大小等。处理模拟信号的计算

机称为模拟计算机，只应用在极特殊的领域中。用模拟信号表示数据的大小有许多缺点，如表示的精度低、表示的范围小、抗干扰能力弱、不便于存储等。如果用数字信号表示信息则可以克服以上缺点。

数字信号是一种在时间上或空间上断续变化的电信号，如电平信号和脉冲信号。单个电信号一般只取两种状态，如电平的高或低、脉冲的有或无，这样就可以用这两种状态分别表示数字代码 1 和 0，称为二值逻辑。比如，用高电平状态表示 1，低电平状态表示 0；或者用有脉冲的状态表示 1，无脉冲的状态表示 0。用 1 位数字信号表示 1 位数字代码，用多位数字信号的组合就可以表示多位数字代码。处理数字信号的计算机称为数字计算机，电平信号和脉冲信号是数字计算机中最基本的电信号形式。用数字信号可以表示数字代码，用数字代码又可以表示各种信息，因

而数字计算机能用于各行各业，处理广泛的信息。下面通过两个例子说明如何用多位电信号的组合来表示多位数字代码。

【例 1-7】 用一组电平信号表示 4 位数字代码。

电平信号利用信号电平的高、低状态表示不同的代码，所以电平信号通常需要一段有效维持时间。可以用 4 根信号线分别输出 4 个电平信号，每个电平信号表示 1 位代码。我们约定，+5 V 为高电平，表示 1；0 V 为低电平，表示 0。如图 1-3 所示，4 位电平信号表示 4 位数字代码 1011，它们可能表示一个 4 位的二进制数，也可能表示一个命令或一种状态的编码。

每一位信号各占用一根信号线，因而这一组电平信号在空间上的分布是离散的。在计算机中常用电平信号表示并行传送的信息，如用若干根信号线同时传送的数据、地址或其他信息的编码。

【例 1-8】 用一串脉冲信号表示 4 位数字代码。

与电平信号不同，脉冲信号的电平维持时间很短，如信号电平从 0 V 向 +5 V（或 -5 V）跳变，维持极短时间后再回到原来的 0 V 状态。因此，信号出现时其电平为 +5 V（或 -5 V），信号未出现时其电平为 0 V，如图 1-4 所示。由于脉冲信号在时间上的分布是离散的，因而可以用一根信号线发出一串脉冲信号，在约定的时间内有脉冲表示 1，无脉冲表示 0。图 1-4 中的脉冲串表示 4 位数字代码 1011。

可以用脉冲信号的上升边沿或下降边沿表示某一时刻，对某些操作定时。例如，在脉冲上升边沿将数据送入某个寄存器中。另外，在计算机中常常用脉冲信号表示串行传送的数据。

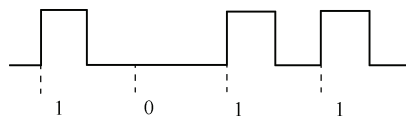


图 1-4 用一串脉冲信号来表示多位数字代码

3. 用数字化方法表示信息的主要优点

（1）在物理上容易实现信息的表示与存储

每一位信号只取两种可能的状态表示 1 或 0，因而在物理上可以用多种方法来实现，如开关的接通或断开、晶体管的导通或截止、电容上有电荷或无电荷、磁性材料的正向磁化或反向磁化、磁化状态的变化或不变等。凡是具有两种稳定状态的物理介质均可用来存储信息，如用双稳态触发器存储信息，或利用电容上存储的电荷来存储信息，还可以用磁性材料记录信息，或者用激光照射过的介质记录信息。

（2）抗干扰能力强，可靠性高

由于单个数字信号的两种状态（高电平与低电平，或者有脉冲与无脉冲）差别较大，即使信

号受到一定程度的干扰，仍然能比较可靠地鉴别出电平的高低或信号的有无。例如，高电平+5 V 表示 1，低电平 0 V 表示 0，假设信号处于 0 状态，如果出现了 2 V 的干扰信号，也不会将原来信号的 0 状态改变到 1 状态。

(3) 数值的表示范围大，表示精度高

一位数字信号的表示范围很窄，但用多位数字信号的组合表示一个数时，可以获得很大的表示范围和很高的精度。例如，用 4 位电平信号表示一个 4 位的二进制整数时（不考虑符号），能够表示的最大数值是 15。若要表示一个 4 位的二进制小数，同样不考虑符号，则数的精度为 2^{-4} 。位数越多，数的表示范围越大，或者数的表示精度越高。从理论上讲，位数的增加是没有限制的，但位数增多，所花费的硬件开销也相应增大。

(4) 可表示的信息类型极其广泛

各种非电量类型的信息可以先转换为电信号，模拟电信号又可以转换为数字电信号，因此表示的信息类型和范围几乎没有限制。

(5) 能用数字逻辑技术进行信息处理

根据处理功能逻辑化的思想，计算机的所有操作最终是用数字逻辑电路来实现的。因此，用逻辑代数对信息进行处理，就形成了计算机硬件设计的基础，可以用非常有限的几种逻辑单元（如与门、或门、非门等）构造出变化无穷的计算机系统和其他数字系统。

从事计算机技术工作的重要基础是善于用约定的数字代码表示各种需要描述的信息。这里再次强调信息数字化这一重要概念：① 计算机中的各种信息都用数字代码表示，这些信息包括数值型的数字、非数值型的字符、图像、声音以及逻辑型的命令、状态等；② 数字代码中的每一位都用数字信号来表示，数字信号可以是电平信号或脉冲信号。

1.1.2 存储程序工作方式

存储程序是计算机的核心内容，表明了计算机的工作方式，包含三个要点：事先编制程序，存储程序，自动、连续地执行程序。这三点体现了用计算机求解问题的过程，下面分别加以说明。

(1) 根据求解问题事先编制程序

计算机处理任何复杂的问题都是通过执行程序来实现的。因此，在求解某一问题时，用户要根据解决这一问题所采用的算法事先编制程序，规定计算机需要做哪些事情，按什么步骤去做。程序中还应提供需要处理的数据，或者规定计算机在什么时候、什么情况下从输入设备取得数据，或向输出设备输出数据。

(2) 事先将程序存入计算机中

如前所述，用户用某种编程语言编写的程序称为源程序，它是由字符组成的，计算机不能识别。因此，需要通过编译器将源程序转换为二进制代码，保存在存储器中。这时的程序还不是指令代码，不能被计算机执行，还需进一步转换为机器指令序列。所以，事先编写的程序最终将变为指令序列和原始数据，并被保存在存储器中，提供给计算机执行。

(3) 计算机自动、连续地执行程序

程序已经存储在计算机内部，计算机被启动后，不需要人工干预，就能自动、连续地从存储器中逐条读取指令，按指令要求完成相应操作，直到整个程序执行完。当然，在某些采用人机对话方式工作的场合，也允许用户以外部请求方式干预程序的运行。

指令和数据都是以二进制代码的形式存放在存储器中的，那么计算机如何区分它们？又如何自动地从存储器中读取指令呢？首先，将指令和数据分开存放。由于在多数情况下，程序是顺序

执行的，因此大多数指令需要依次相邻存放，而将数据放在该程序区中不同的区间。其次，可以设置一个程序计数器（Program Counter, PC），用它存放当前指令所在的存储单元的地址。如果程序顺序执行，则在读取当前指令后将 PC 的内容加 1（当前指令只占用一个存储单元），指示下一条指令的地址。如果程序要进行转移，则将转移目标地址送入 PC，以便按照转移地址读取后续指令。所以，依靠 PC 的指示，计算机就能自动地从存储器中读取指令，再根据指令提供的操作数地址读取数据。

对于传统的冯·诺依曼机而言，存储程序工作方式是一种控制流驱动方式，即按照指令的执行序列依次读取指令，再根据指令所含的控制信息调用数据进行处理。这里的控制流也称为指令流，是指在程序执行过程中，各条指令逐步发出的控制信息，它们始终驱动计算机工作。而依次被处理的数据信息称为数据流，它们是被驱动的对象。

1.1.3 计算机的分类

计算机的种类有多种多样，从不同的角度出发，可以将它们分成不同的种类。

首先，按计算机中处理信息的制式，计算机可以分为数字计算机和模拟计算机。

数字计算机是通过信号的两种不同状态来表示数字信息（1 和 0）的，可以方便地对数字信号进行算术和逻辑运算，它具有速度快、精度高、便于存储等优点。目前，通常讲的计算机，一般都是指数字计算机。

模拟计算机一般只能处理模拟信号，如连续变化的电压、电流和温度等，主要由模拟信号运算器件组成，适合于求解微分方程等。这种计算机在模拟计算和实时控制系统中有应用，但通用性不强，信息不易表示和存储，精度也不高，因此它的应用范围很窄。

其次，如果按照计算机通用性的差异，计算机可以分为专用计算机和通用计算机。

专用计算机，即专门为解决某一特定问题而设计制造的计算机。它一般只具有为实现某一个特定任务而定制的最小软硬件结构，一般也拥有相对固定的存储程序。专用计算机具有执行速度快、可靠性高、结构简单和价格便宜等优点，但它的功能单一，对应用环境的适应性很差。例如，控制轧钢过程的轧钢控制计算机，计算导弹弹道的专用计算机等，都属于专用计算机的范畴。

通用计算机指各行业和各种工作环境都能使用的计算机，一般具有较高的运算速度、较大的存储容量、配备了比较齐全的外部设备及软件。与专用计算机相比，为了实现其通用性，通用计算机的结构比专用计算机更复杂、价格也更昂贵。

通用计算机的适应能力很强，其应用范围也很广。它的运行效率、速度和经济性等指标在不同的应用场景下表现会有很大差别。我们平常使用的个人台式计算机等都属于通用计算机。

对于典型的通用计算机，如果按照计算机系统的规模和处理能力等技术指标，从小到大、从弱到强地进行区分，大致上可以分为如下几类。

（1）微型机

微型机，也俗称微机，是计算机领域中目前发展得最快、应用也最广泛的一种计算机。第一台典型意义上的微机 MCS-4，是由美国的 Intel 公司在 1971 年以自主研发的 4 位微处理器（Intel 4004）为基础，扩展并增加存储系统和输入/输出接口等后形成。从那以后，Intel、MITS 和 IBM 等公司又陆续推出了更新型的微机产品。

常见的微型机有个人台式计算机、笔记本电脑、一体机和工作站等。

微型计算机和与其他类型计算机的主要区别在于，微型计算机广泛采用了集成度相当高的电子元件和独特的总线（Bus）结构。除此之外，微型计算机还具有轻便、小巧、价格低、操作和

使用都很方便等特点，其应用范围最广，发展普及也最快，现在已经成为大众化的信息处理和数字娱乐工具。

（2）小型机

小型计算机是相对于大型计算机而言的，小型计算机的软件、硬件系统规模比较小，但价格低、可靠性高、便于维护和使用。

小型机最初是在 20 世纪 70 年代由美国 DEC 公司首先开发的一种高性能计算产品，曾经风行一时。小型计算机也曾用来表示一种多用户、采用主机/终端模式的计算机，它的规模和性能介于大型计算机和微型计算机之间。目前主流小型机的内部一般都集成了几十或上百个 CPU，且采用不同版本的 UNIX 操作系统，常用作为中高端的专业服务器。国内的服务器领域还习惯性地将各类 UNIX 服务器简称为小型机。

小型机采用的是主机/终端模式，并且各家厂商均有各自的体系结构，如处理器架构、I/O 通道和操作系统软件等都是特别设计的，一般彼此之间互不兼容。此外，与普通服务器相比，小型机还具有高 RAS (Reliability, Availability, Serviceability) 特性，即：高 Reliability (可靠性)，计算机可以 7×24 持续工作永不停机；高 Availability (可用性)，重要资源都有备份，能检测到潜在异常，能转移任务到其他资源以减少停机时间保持持续运行，且具备实时在线维护和延迟性维护等功能；高服务性 (Serviceability)，能够实时在线诊断，精确定位发生的故障，并能做到准确无误的快速修复。

（3）大型机

大型计算机简称大型机，一般作为大型的高性能商业服务器，因其一般具有较大的体积（通常占地面积几十平方米）而得此名。

大型机通常使用专用的处理器指令集（如 IBM 公司的 Z/Architecture 架构 CISC 指令集）、专用的操作系统（如 Z/OS）和专用的应用软件（如 IBM DB2 数据库系统），通常具有较高的运算速度，一般为每秒钟数亿次级别，还具有较大的存储容量，具备较好的通用性，功能也比较完备，能支持大量用户同时使用计算机数据和程序，具有强大的数据处理能力，但大型机的价格也比较昂贵。

大型机除了像小型机那样要求高 RAS 性能外，还特别强调了 I/O 数据的吞吐率和处理器指令架构的兼容性。大型机一般通过专用处理器来控制通道进行 I/O 处理，一个 I/O 通道能同时处理多个 I/O 操作、控制上千个 I/O 设备，因此能同时处理上千个数据流，还能保证每个数据流高速运转。大型机的高 RAS (可靠性、可用性、服务性)、分区和负载能力等及 I/O 性能优势是其他类型服务器所不能匹敌的。大型机处理复杂的多任务时能表现出超强的处理能力，其宕机时间也远远低于其他类型的服务器。大型机 I/O 能力强，擅长超大型数据库的访问，采取动态分区管理，根据不同应用负载量的大小，灵活地分配系统资源；从底层防止入侵的设计策略使大型机安全性提高。

目前，能生产大型机的企业主要有美国的 IBM 和 UNISYS 等公司，其中 IBM 公司生产的大型机系列产品几乎占据了全球 90% 以上的市场份额，曾于 2005 年与电子科大签署合作协议，并提供一台 IBM eServer Z900 大型机用于教学 and 培训。大型机通常应用在银行、证券和航空等大型企业中对大数据处理能力和系统的安全性、稳定性等都有极为苛刻要求的应用场合。我国仅有中科院计算所、国防科大和浪潮等单位能设计和生产大型计算机。

（4）超级计算机

超级计算机，早期叫巨型机，现在常简称为“超算”。与大型机相比，超级计算机通常由成

千上万个计算节点和服务节点组成，具有更强大的计算和处理数据的能力，主要特点表现为超高的计算速度和超大的存储容量，并配有多种外部和外围设备及多种功能丰富的软件系统。

超级计算机是计算机中功能最强、运算速度最快、存储容量最大的一类计算机，多用于国家高科技领域和尖端技术研究，是一个国家科研实力的体现，它对国家安全，经济和社会发展具有举足轻重的意义，是国家科技发展水平和综合国力的重要标志。

超级计算机和大型机的主要区别有如下几点：

① 大型机使用专用指令系统和操作系统，而超级计算机使用通用处理器及 UNIX 或类 UNIX 操作系统，如 Linux 等。

② 大型机擅长非数值计算（数据处理），而超级计算机擅长数值计算（科学计算）。

③ 大型机主要用于商业领域，如银行和电信等，而超级计算机常用于尖端科学领域，特别是国防和天气预报等领域。

④ 大型机大量使用冗余等技术确保其安全性及稳定性，所以内部结构通常会有备份。而超级计算机使用大量的处理器，通常由多个机柜组成，体积比大型机更大。

超级计算机的主机主要由高速运算部件和大容量快速主存储器等部件构成。由于超级计算机加工数据的吞吐量比大型机更大，除了主存外一般还有半导体快速扩充存储器和海量（磁盘）存储子系统来支持。超级计算机的主机一般不直接管理低速 I/O 设备，而是通过 I/O 接口通道连接前端机，一般是小型机，由前端机处理 I/O 任务。此外，I/O 的另一种途径是通过网络，联网用户借助其终端机（微型、小型或大型机）通过网来与超级计算机交互，I/O 均由用户终端机来完成，这种方式可大大提高超级计算机的利用率。

为提高系统性能，现代的超级计算机都在系统结构、硬件、软件、工艺和电路等方面采取各种支持并行处理的技术。例如，一般都采用多处理机结构，且处理器除了支持传统的标量数据外，还增加了向量或数组类型数据；硬件方面大多都采用流水线、多功能部件、阵列结构或多处理机、向量寄存器、标量运算、并行存储器等多种先进技术。

我国的超级计算机研制开始于 20 世纪 60 年代，国防科大慈云桂教授主持研发的国内首台超级计算机“银河-I”于 1983 年 12 月 22 日诞生，使我国继美国、日本之后成为第三个高性能计算机研制生产国。中国现阶段超级计算机的拥有量超过 60 台，代表性的有国防科大研制的“天河”系列以及中科院研制的“曙光”系列超级计算机，无论拥有量还是运算速度，我国在世界上都处于领先的地位。

注意，计算机领域中对于微型机、小型机、大型机和超级计算机的划分都是一个相对的概念，其划分标准也会随着技术的不断发展而发生动态变化。正因为如此，目前的微型机，其计算性能或许相当于数十年前的小型机，又或者现在的小型机，若干年之后，说不定它还赶不上那时微型机的综合性能指标。

1.2 计算机的诞生和发展

1.2.1 冯·诺依曼体系

计算机系统作为一个能够自动地处理信息的智能化工具，必须解决好两个最基本的问题：第一，信息如何表示，才能方便地让计算机识别和处理；第二，采用什么工作方式，才能使计算机自动地对信息进行处理。

对上述两个问题的解决方案做出杰出贡献并产生深远影响的是一位美籍匈牙利科学家约

翰·冯·诺依曼 (John von Neumann, 1903—1957), 如图 1-5 所示。冯·诺依曼在 1944 年加入了美国军方 ENIAC (Electronic Numerical Integrator And Computer) 计算机的研制项目。在此研制基础之上, 他在 1945 年提出并发表了一个全新的“存储程序通用电子计算机”方案——EDVAC (Electronic Discrete Variable Automatic Computer)。在这份方案中, 冯·诺依曼具体介绍了制造电子计算机和程序设计的新思想, 这份方案是计算机发展史上一个划时代的文献, 它向世界宣告了电子计算机时代的来临。

EDVAC 方案明确奠定了新型计算机应由 5 部分组成: 运算器、控制器、存储器、输入设备和输出设备, 并描述了这 5 部分的功能和相互关系。该方案还对 EDVAC 的两大设计思想做了进一步论证, 为新型电子计算机的设计树立了一座里程碑, 具有划时代的意义。

设计思想之一是信息采用二进制来表示。他根据电子元件的双稳态工作特点, 建议在电子计算机中采用二进制来表示信息, 他还分析了二进制的优点, 并预言二进制的采用将大大简化计算机硬件系统的逻辑结构。

设计思想之二是程序存储的思想。把运算程序存在计算机的存储器中, 程序设计员只需要在存储器中寻找运算指令, 计算机就能自行计算, 这样就不必对每个问题都重新编程, 从而大大加快了运算的进程。程序存储的思想标志着计算机期望的自动运算的实现, 同时标志着电子计算机设计思想的成熟, 且已成为电子计算机设计的基本原则。

总体上, 冯·诺依曼体系的主要思想可概括为: ① 计算机硬件系统由五大部件 (存储器、运算器、控制器、输入设备和输出设备) 组成; ② 计算机中采用二进制形式表示信息 (数据、指令); ③ 采用存储程序的工作方式, 这也是冯·诺依曼体系最为核心的思想。

冯·诺依曼针对 EDVAC 提出的新型电子计算机体系结构及设计思想, 奠定了现代电子计算机设计的理论基础, 并开创了程序设计的新时代。因此, 他提出的这些计算机设计思想也被称为冯·诺依曼思想, 相应的计算机体系结构也被称为冯·诺依曼体系结构, 也正是因为对计算机的这种划时代贡献, 他被后世尊称为“计算机之父”。

采用冯·诺依曼思想体制的计算机就称为冯·诺依曼计算机。近几十年来, 尽管计算机的体系结构已经发生了许多演变, 但它们总体上仍然没有脱离冯·诺依曼体制的核心思想原则, 现在的绝大多数计算机仍然属于冯·诺依曼机体系结构的计算机。当然, 从本质上讲, 传统的冯·诺依曼机采用的是串行处理的工作机制、逐条执行指令序列。要想提高计算机的性能, 其根本方向之一是采取并行处理机制, 如用多个处理部件形成流水线, 依靠时间上的重叠来提高处理效率, 或者用多个冯·诺依曼机组成多机系统, 以支持并行算法等。



图 1-5 约翰·冯·诺依曼

1.2.2 计算机发展历程

1946 年 2 月 14 日, 在美国的宾夕法尼亚大学, 世界上第一台严格意义上的电子数字计算机 ENIAC (Electronic Numerical Integrator and Calculator, 电子数字积分器与计算器) 诞生了, 它标志着电子计算机时代的到来。这台计算机是由美国军方定制, 专门为了计算导弹的弹道和射击特性而研制的, 承担开发任务的“莫尔小组”主要由四位科学家和工程师埃克特、莫克利、戈尔斯坦、博克斯组成, 后来在研制过程中, 冯·诺依曼作为技术顾问也加入了该小组。这台计算机主要元器件采用的是电子管, 共使用了 1500 个继电器, 18800 个电子管, 占地约 170 m², 重达 30 多吨, 耗电 150 kW, 造价高达 48 万美元, 每秒能完成 5000 次加法运算, 400 次乘法运算, 比当

时最快的计算工具快 300 倍，是继电器计算机的 1000 倍、手工计算的 20 万倍。

第一台电子计算机于 1946 年诞生后，发展到现在已经历了几十年，也经历了不同的技术发展阶段，且每个阶段也都各具特色。

1. 第一代，电子管计算机（1946—1957）

这一阶段计算机的主要特征是采用电子管元件作基本器件，一般使用光屏管或汞延时电路作为存储器，输入或者输出主要采用穿孔卡片或纸带等，通常体积比较大、功耗较高、计算速度较慢、存储容量较小、可靠性比较差、维护也很困难而且价格很昂贵。在程序设计上，通常使用机器语言或者汇编语言来编写程序，这一代计算机主要用于军事科学计算。

电子计算机在我国的研制起步较晚。1957 年 4 月，我国经政府途径从苏联订购了 M-3 计算机的技术资料，在这些技术资料的基础上开始研制。以中科院计算技术研究所研究员张梓昌和莫根生为主，组织了 M-3（代号为 103）计算机工程组。通过与北京有线电厂的密切配合，工程组于 1958 年 8 月 1 日成功研制（实为在苏联专家指导下仿制）我国第一台小型电子管数字计算机——103 型机（定点 32 位，最初的平均运算速度仅为每秒 30 次），填补了国内电子计算机的空白。后经技术改进，其平均速度提高到每秒 1800 次，定名为 DJS-1 型计算机。

在苏联专家指导下，以 Б Э С М-II 机为蓝本，1959 年 10 月，我国宣布研制成功了第一台大型电子管数字通用计算机（浮点 40 位，平均速度每秒达 1 万次）即 104 型机，后定名为 DJS-2 型计算机，该机的技术指标在当时不亚于英国和日本的水平。

103 机和 104 机属于我国的第一代电子管计算机，它们的相继推出标志着我国初生的计算机事业终于蹒跚起步，为我国解决了大量过去无法计算的经济和国防等领域中的计算难题，填补了我国计算机技术的空白，是我国计算机工业发展史上的第一个里程碑。

在研制 104 机的同时，中科院计算所夏培肃研究员领导的小组首次自行设计并于 1960 年 4 月研制成功一台小型的串行通用电子管数字计算机 107 机（定点 32 位，平均每秒 250 次）。1964 年，中科院计算所吴几康研究员等主持的我国第一台自行设计的大型通用电子管计算机 119 机（浮点 44 位，平均每秒 5 万次）也研制成功。

2. 第二代，晶体管计算机（1958—1964）

20 世纪 50 年代中期，晶体管的出现使计算机生产技术得到了根本性的发展，晶体管代替了电子管作为计算机的基础器件，普遍采用磁芯和磁鼓作为存储器。在整体性能上，它比第一代计算机有了很大的提高，速度更快、寿命更长、体积更小、重量更轻且功耗更低。同时，许多专用的计算机程序设计的高级语言也相应出现了，如 FORTRAN、COBOL 和 Algo160 等。在此阶段，晶体管计算机在被用于科学计算的同时，也开始在数据处理、过程控制等领域得到了广泛应用。

我国的晶体管计算机发展也远落后于世界发达国家，在研制电子管计算机的同时，也在着手晶体管计算机的研制。国内在 1963 年开始才出现晶体管计算机，如“109 原型机”和“441B”等。直到 1965 年，我国第一台独立设计研制的大型晶体管计算机“109 乙”（浮点 32 位，每秒 6 万次）才由中科院计算所正式研制成功，标志着我国才开始正式进入了第二代（晶体管）计算机时代。此时，国外已在大力研制第三代计算机了。后经过对“109 乙”历时两年的技术改进，又成功推出了“109 丙”计算机，该机为“两弹一星”实验做出了卓越贡献，因此被誉为“功勋计算机”。国内其他科研院所还陆续推出了 108 机、121 机和 320 机等型号的晶体管计算机。

3. 第三代，中小规模集成电路计算机（1965—1971）

20 世纪 60 年代中期，随着半导体工艺的发展，集成电路开始出现。在此阶段，中小规模集

成电路逐渐成为计算机的主要逻辑部件,半导体存储器也开始采用,这些改进使计算机体积更小、功耗更低,计算速度和可靠性更高。在软件方面,有了标准化的程序设计语言和人机会话式的 BASIC 语言,程序设计方法上也出现了结构化程序设计思想,为编制更复杂的软件提供了技术保证。与此同时,逐渐出现了分时操作系统,多用户可共享计算机软件、硬件资源,这些变化都导致计算机的应用领域进一步扩大。

我国对第三代计算机的研制受到“文革”的巨大冲击,整体上要落后国外一代。1964 年,美国 IBM 公司就推出了 360 系列大型机,那时我国尚在研制第二代计算机,第三代计算机的研制尚在规划阶段。1972 年,我国才研制成功运算速度达到每秒 11 万次的大型集成电路通用计算机。1973 年,北京大学与北京有线电厂合作,才研制成功了我国第一台运算速度在百万次级的大型集成电路通用计算机“150 机”(支持浮点 48 位运算,计算速度达到每秒 100 万次)。

进入 20 世纪 80 年代,我国的高速计算机特别是向量计算机技术有了新的发展。1983 年,中科院计算所研制完成我国第一台大型向量计算机“757 机”问世,其计算速度可达每秒 1000 万次,这一速度记录同年 11 月又被国防科大研制的巨型计算机“银河-I”打破,其计算速度可达每秒 1 亿次。银河-I 巨型机是我国高速计算机研制的一个重要里程碑,它的研制成功表明中国成了继美、日等国之后,能够独立设计和制造巨型机的国家。

4. 第四代,大规模和超大规模集成电路计算机(1971 年至今)

大规模和超大规模集成电路(VLSI)技术逐渐成熟,因此在计算机中普遍采用大规模和超大规模集成电路作为核心逻辑部件,这使得计算机的体积、功耗和成本等更进一步降低,微型计算机也随之出现。除此此外,半导体存储器的集成度越来越高,容量也越来越大,还发展出了并行技术和多机系统,出现了精简指令集计算机(RISC),软件系统工程化、理论化和程序设计自动化也得到充分发展。软盘、硬盘和光盘及 U 盘等辅助存储器也相继出现,各种新颖的输入、输出设备不断推陈出新,软件产业也高速发展,各类应用软件层出不穷,计算机与通信技术相结合的典范——计算机网络也得到了迅猛发展,多媒体技术也在这个阶段异军突起。微型计算机在社会上的应用和普及范围进一步扩大,几乎所有领域都能看到计算机的存在。

与国外一样,我国的超大规模集成电路计算机的研制也是从微机开始的,只不过从 20 世纪 80 年代中期才开始。20 世纪 80 年代初期,国内就有很多单位开始采用 Z80、X86 和 M6800 等芯片研制微机。1985 年,原电子部六所研制成功能与 IBM PC(1981 年)兼容的 DJS-0520CH 微机,随后在 1987 年第一台国产 286 微机长城 286 问世。

1995 年,曙光 1000 大型机通过鉴定,其峰值可达每秒 25 亿次。曙光 1000 与美国 Intel 公司 1990 年推出的大规模并行机体系结构与实现技术相近,此时与国外先进技术之间的差距缩小到 5 年左右。2000 年,我国自行研制成功高性能计算机“神威 I”,其主要技术指标和性能达到国际先进水平。我国成为继美国、日本之后成为世界上第三个具备自行研制高性能计算机能力的国家。

2001 年,中科院计算所研制成功我国第一款通用 CPU 芯片“龙芯”。2002 年,曙光公司推出完全自主知识产权的“龙腾”服务器,龙腾服务器采用了“龙芯-1”CPU,采用了曙光公司和中科院计算所联合研发的服务器专用主板,采用曙光 Linux 操作系统,该服务器是国内第一台完全实现自有产权的计算机产品,在国防等部门发挥了重大作用。2009 年,国防科技大学研制成功“天河一号”超级计算机,峰值速度达到 1.206 PFLOPS,实测 563.1 TFLOPS,为国内首台达到千万亿次运行速度的超级计算机,其改进版本“天河一号 A”上搭载了 2048 颗国产 FT-1000 处理器(8 核 64 线程),在架构设计上采用了创新的系统架构设计,达到了很高的计算速度,其实测速度达到 2.566 PFLOPS。

目前,在高性能计算机领域,无论是超级计算机的拥有量还是运算速度,我国在世界上都处于领先地位,其中最具代表性的是国防科大研制的“天河”系列和中科院研制的“曙光”系列超级计算机,其最高速度已突破每秒千万亿次,如“天河二号”内置 4096 颗国产 FT-1500 处理器(16 核 SPARC V9 架构,40 nm 制程,主频 1.8 GHz),持续计算速度高达 33.86 PFLOPS(千万亿次 64 位浮点数运算),使其在“国际 TOP500 组织”2013 年 11 月 18 日公布的排行榜中力压美国能源部下属橡树岭国家实验室的“泰坦”名列第一。

1.2.3 未来的发展趋势

自从 1946 年诞生后,计算机经过了四个发展阶段,核心器件从最初的电子管、晶体管再发展到后来的集成电路和超大规模集成电路,其种类越来越多,功能和性能越来越强大,应用范围也越来越广,几乎涉及社会的方方面面。随着科学技术的不断发展,计算机也持续不断地向前发展。从总体上看,计算机的发展发展趋势可以简单地用五个字来加以概括,即“巨”、“微”、“多”、“网”和“智”。

1. 巨型化

计算机的巨型化是指发展运算高速、大容量存储和功能强大的超大型计算机。这既是军事、天文、气象、原子和核反应等尖端科学领域发展的迫切需要,也是人类进一步探索新兴科学领域,诸如宇宙工程、生物生命工程等的需要,也是为了让计算机能具有像人脑一样高度发达的学习和推理等复杂功能。尤其是在信息爆炸性增长的时代背景之下,海量数据的记忆、存储和处理是必要的,这也会促进未来的计算机朝巨型化的方向发展。

2. 微型化

计算机的微型化与巨型化刚好相反。正是因为大规模和超大规模集成电路技术的不断成熟,计算机的微型化发展十分迅速。微型机可以被集成到诸如仪表、家用电器等中小型计算机无法进入的领域,所以自 20 世纪 80 年代以来微型计算机的发展异常迅速。随着微型化的发展,计算机的性能指标进一步提高,其价格也在逐年下降。当前的主流微机的显著标志是将运算部件和控制部件集成在一起,今后微型化的不断深入,将逐步发展到对存储器、通道处理机、高速运算部件、各种接口甚至外围设备等的高度集成,进一步将系统软件固化,从而达到整个微机系统的集成。

3. 多媒体化

计算机的多媒体化是指计算机与以数字技术为核心的图文声像和通信技术等融合在一起发展。计算机多媒体化的目标是:无论何时何地,只需要简单的设备,就能自由地以交互和对话的方式交流信息,其实质是让人们利用计算机以更加自然、简单的方式进行交流,提供更直观的交互方式、更好的用户体验。

4. 网络化

计算机网络是计算机技术发展中崛起的又一重要分支,是现代通信技术与计算机技术结合的产物。从单机走向联网,是计算机应用发展的必然结果。所谓计算机网络,就是在一定的地理区域内,将分布在不同地点的不同机型的计算机和专门的外部设备由通信线路互连在一起,组成一个规模大、功能强的网络系统,在网络软件的协助下,以共享信息、共享软件、硬件和数据资源。计算机网络最初于 1969 年在美国建成,从阿帕网(ARPANET)开始,已迅速发展成为今天体量庞大、跨越全球的互联网,它把不同的国家、地区和个人紧密联系在一起。

计算机的网络化，既是指计算机系统的发展必然要能支持多种网络协议和网络接口，也强调了通过网络把数量众多的计算机和各类终端设备整合在一起，形成一台虚拟的超级计算机，从而实现对计算资源、存储资源、数据资源、信息资源、知识资源、专家资源的深度共享。

5. 智能化

智能化是指让计算机模拟人的感觉、行为和思维等过程，从而使计算机具备和人一样的思维和行为能力，形成智能型和超智能型的计算机。智能化的研究包括模式识别、物形分析、自然语言的生成和理解、定理的自动证明、自动程序设计、专家系统、学习系统、智能机器人等。人工智能的研究使计算机远远突破了“计算”的最初含义，从本质上拓宽了计算机的能力，可以更好地代替或超越人的脑力劳动，如神经网络计算机。

计算机的发展趋势除了向巨型化、微型化、多媒体化、网络化和智能化方向发展以外，目前还处于研制阶段的采用光器件的光子计算机和生物器件的生物计算机是迄今为止最新的一代计算机，这些计算机从本质上已经超越了“电子计算机”的含义。生物计算机的存储能力巨大，处理速度极快，能量消耗极微，总体具有模拟人脑的能力。光子计算机利用光子代替电子，利用光互连代替导线互连，以光器件代替电子器件，以光运算代替电子运算，理论上比传统的电子计算机具有更高的计算速度、更低的功耗以及更好的可靠性。

电子计算机，作为人类文明发展进程中的一种重要工具，既然有它的发生和发展，也必然会有它的衰落直至消亡，这是任何事物的规律。或许，也仅仅是或许，下一代的新型计算机就是生物计算机或者光子计算机。

1.3 计算机系统的组织

1.3.1 硬件系统

在冯·诺依曼体制中，计算机硬件系统是由存储器、运算器、控制器、输入设备和输出设备五大部件组成的。随着计算机技术的发展，计算机硬件系统的组织结构已发生了许多重大变化，如运算器和控制器已组合成一个整体，称为中央处理器（Central Processing Unit, CPU），存储器已成为多级存储器体系，包含主存、高速缓存和外存三个层次。下面先以目前常见的计算机硬件系统组成为例，讨论系统中各部件应该具有哪些功能，以及这些部件通过什么方式相互连接构成整机等硬件设计方面的问题，然后简单介绍典型的硬件系统结构。

1. 计算机硬件系统基本组成

图 1-6 是一种计算机硬件系统的简化结构模型示意图，其中包含 CPU、存储器、输入/输出（I/O）设备和接口等功能部件，各部件之间通过系统总线相连接。

（1）CPU

CPU 是计算机硬件系统的核心部件，在微型计算机系统或其他应用大规模集成电路技术的系统中，它被集成在一块芯片上，构成微处理器。CPU 的主要功能是读取并执行指令，在执行指令的过程中，它向系统中的各部件发出各种控制信息，收集各部件的状态信息，与各部件交换数据信息。

CPU 由运算部件、寄存器组和控制器组成，它们通过 CPU 内部的总线相互交换信息。运算

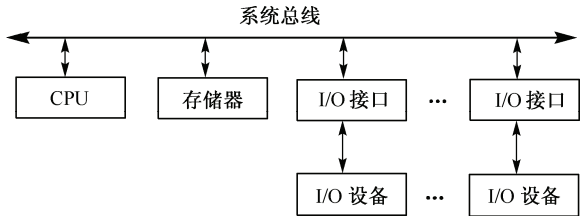


图 1-6 计算机硬件系统的结构模型

部件完成算术运算（定点数运算、浮点数运算）和逻辑运算。寄存器组用来存放数据信息和控制信息。控制器提供整个系统工作所需的各种微命令，这些微命令可以通过组合逻辑电路产生，也可以通过执行微程序产生，相应分别被称为组合逻辑和微程序控制方式。

（2）存储器

存储器用来存储信息，包括程序、数据、文档等。如果存储器的存储容量越大、存取速度越快，那么系统的处理能力就越强，工作速度也就越高。但是一个存储器很难同时满足大容量、高速度的要求，因此常将存储器分为主存、外存和缓存三级存储体系。

主存用来存放 CPU 需要使用的程序和数据。主存的每个存储单元都有固定的地址，CPU 可以按地址直接访问它们。因此，要求主存的存取速度很快，但目前因技术条件的限制其容量有限，一般仅为几 GB。主存通常用半导体材料构成。此外，通常将 CPU 和主存合称为主机，因主存位于主机之内，故主存又常被称为内存。

外存位于主机之外，用来存放大量的需要联机保存但 CPU 暂不使用的程序和数据。需要时，CPU 并不直接按地址访问它们，而是按文件名将它们从外存调入主存。因此，外存的容量很大，但存取速度比主存慢，如磁盘、光盘和 U 盘等都是常用的外存。

高速缓存（Cache）是为了提高 CPU 的访存速度，在 CPU 和主存之间设置的一级速度很快的存储器，容量较小，用来存放 CPU 当前正在使用的程序和数据。高速缓存的地址总是与主存某一区间的地址相映射，工作时 CPU 首先访问高速缓存，如果未找到所需的内容，再访问主存。高速缓存由高速的半导体存储器构成。在现代计算机中，缓存是集成在 CPU 内部的，一般集成了两级 Cache，高端芯片（如多核处理器）甚至集成了第三级 Cache。早期的计算机中常在 CPU 外部设置片外 Cache，但这种方式已经被淘汰很多年了。

（3）输入/输出设备

输入设备将各种形式的外部信息转换为计算机能够识别的代码形式送入主机。常见的输入设备有键盘、鼠标等。输出设备将计算机处理的结果转换为人们所能识别的形式输出。常见的输出设备有显示器、打印机等。

从信息传送的角度来看，输入设备和输出设备都与主机之间传输数据，只是传输方向不同，因此常将输入设备和输出设备合称为输入/输出（Input/Output, I/O）设备。它们在逻辑划分上位于主机之外，又称为外围设备或外部设备，简称外设。磁盘、光盘等外存既可看成存储系统的一部分，也可看成具有存储能力的输入/输出设备。

（4）总线

总线是一组能为多个部件分时共享的信息传输线。现代计算机普遍采用总线结构，用一组系统总线将 CPU、存储器和 I/O 设备连接起来，各部件通过这组总线交换信息。注意：任意时刻只能允许一个部件或设备通过总线发送信息，否则会引起信息的碰撞；但允许多个部件同时从总线上接收信息。

根据系统总线上传送的信息类型，系统总线可分为地址总线、数据总线和控制总线。地址总线用来传送 CPU 或外设发向主存的地址码。数据总线用来传送 CPU、主存以及外设之间需要交换的数据。控制总线用来传送控制信号，如时钟信号、CPU 发向主存或外设的读/写命令和外设送往 CPU 的请求信号等。

（5）接口

在图 1-6 中，为什么在系统总线与 I/O 设备之间设置了接口部件，如 USB 接口、SATA 接口和 PCI-E 接口等？这是因为计算机通常采用确定的总线标准，每种总线标准都规定了其地址线和

数据线的位数、控制信号线的种类和数量等。但计算机系统所连接的各种外部设备并不是标准的，在种类与数量上都是可变的。因此，为了将标准的系统总线与各具特色的 I/O 设备连接起来，需要在系统总线与 I/O 设备之间设置一些部件，它们具有缓冲、转换、连接等功能，这些部件就被称为 I/O 接口。

计算机的各种操作都可以归结为信息的传输。信息在计算机中沿着什么途径传输将直接影响硬件系统结构。我们将信息在计算机中的传送途径称为数据通路结构。因此，硬件系统结构的核心是数据通路结构。不同类型的计算机，如传统的微型机、小型机和大、中型机，其功能侧重点不同，因而它们的数据通路结构是有区别的。下面介绍几种典型的计算机硬件系统结构及其特点。

2. 典型的硬件系统结构及其特点

(1) 微型计算机的“南-北”桥经典架构

图 1-7 是一个典型的微型计算机硬件系统架构模型示意图。该模型基于 Intel 平台经典的“南-北”桥布局结构，广泛流行多年。在这种架构模型中，CPU、存储器、输入/输出设备和接口等部件通过各类总线实现互连互通。

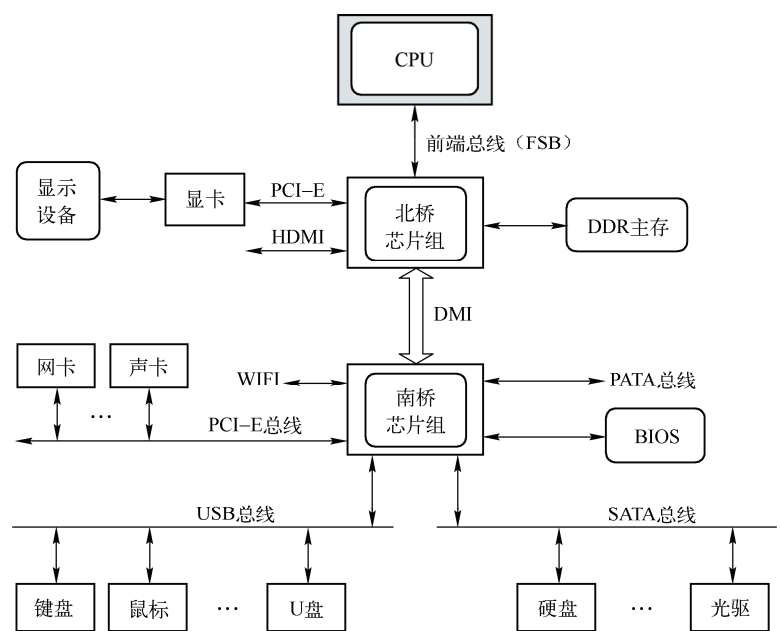


图 1-7 典型的微机硬件系统架构模型

从总体架构上看，图示的微机采用的是南-北桥架构：北桥芯片组（North Bridge Chipset）主要承担内存控制、视频控制和与 CPU 的交互；南桥芯片组（South Bridge Chipset）负责控制外部设备的输入/输出，如键盘、鼠标、硬盘、网络设备等，还承担 BIOS（Basic Input/Output System）的管理任务。北桥（有的资料上也称为主桥）与南桥之间通过 DMI（Direct Media Interface）即直接媒体接口标准的总线相互连接，以形成“CPU-北桥-南桥-外设”这种模式信息传输与控制方式。

图 1-7 中的体系结构模型中可能存在多种不同的总线，如 FSB、DMI、PATA、PCI-E、USB 和 SATA 等。

(2) 小型计算机的硬件体系架构

微型机和小型机系统往往侧重于以较低的硬件代价实现较强的系统功能，因此常用多组系统

总线作为系统中各部件互连的基础，如连接 CPU、存储器和 I/O 接口，再通过接口连接外部设备。图 1-8 展示了惠普公司的一种名为 ProLiant DL300 的系列双处理器小型机（服务器）的硬件系统架构模型。该系列小型计算机采用双 CPU，Intel C600 系列芯片组，且没有再采用经典的“南-北”桥架构布局。与微型计算机经典的“南-北”桥架构布局相比，该小型机仅使用了一个芯片组，实际上可以理解成处理器已经集成了绝大多数北桥芯片的功能，如内存控制和显示核心等，只需保留南桥芯片（Intel C600 系列芯片组），实际上这就是目前主流服务器的单桥体系架构模型。

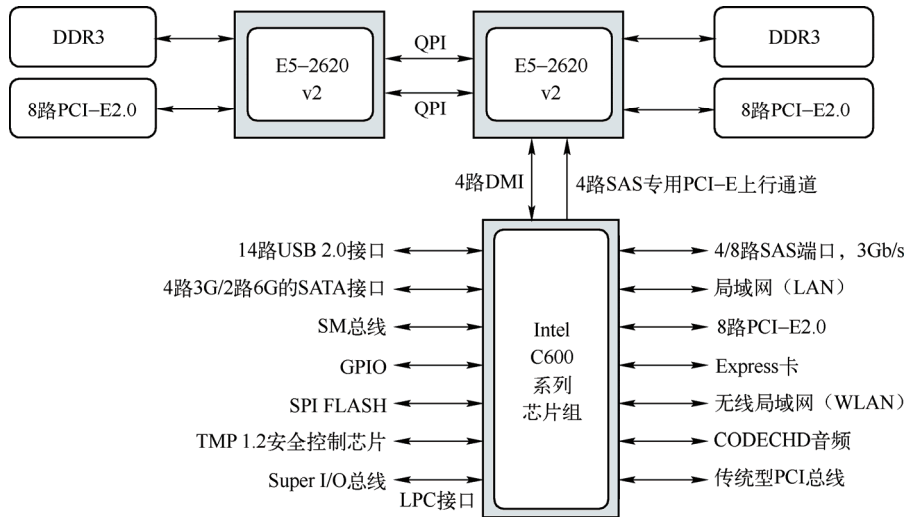


图 1-8 惠普 ProLiant DL300 系列小型机的硬件架构

在这种双处理器机架式服务器中，两个处理器（Intel Xeon E5-2620 v2，22 纳米工艺制程，64 位 Ivy Bridge 微架构，6 核 12 线程，2.1 GHz 主频，共享 15 MB 三级缓存）之间通过 QPI（Quick Path Interconnect）总线互连。主处理器通过 DMI 总线与 Intel C600 系列芯片组互连。此外，芯片组提供 14 路 USB 2.0 接口、SM（系统管理）总线、GPIO（General Purpose Input Output）即通用输入/输出端口、TMP 安全控制芯片、Super I/O 总线、普通 PCI 总线和 4/8 路带宽为 3Gb/s 的 SAS（Serial attached SCSI）磁盘接口，以及 8 路 PCI-E 2.0 总线和 PCI Express 卡插槽、无线局域网接口和传统的音频输出接口等。

Intel C600 系列芯片组提供的接口、插槽、总线能对存储系统和外围设备等进行扩展，构建一个高性能的企业级专用小型计算机服务器。这种体系架构中设置了 2 个处理机，因此也可以看成一个多处理机系统，且两个处理机都有自己的专有存储器，共享主存。这种模式也可以看成一种紧密耦合型多处理机系统。

（3）超级计算机的硬件体系架构

对于计算机的硬件体系架构，除了微型计算机的单处理器、南-北桥架构和小型计算机的多处理器、单桥架构外，常见的还有一种采用多处理机分布式架构的超级计算机体系架构，图 1-9 展示了国防科大研制的 TH-2（天河二号）超级计算机的架构模型。总体架构上，天河二号超级计算机分成 4 个集群：管理集群、计算集群、存储集群和通信集群，各功能集群之间通过 TH Express-2 主干网形成“胖树”拓扑结构互连，传输技术为光电混合传输及专有网络协议（Proprietary Network Protocol）。系统中的通信集群由 13 个大机柜构成，每个机柜安装一台 576 个端口的大型路由器，还有各级交换机若干，各计算节点通过 PCI-E 2.0 标准接入到通信集群。

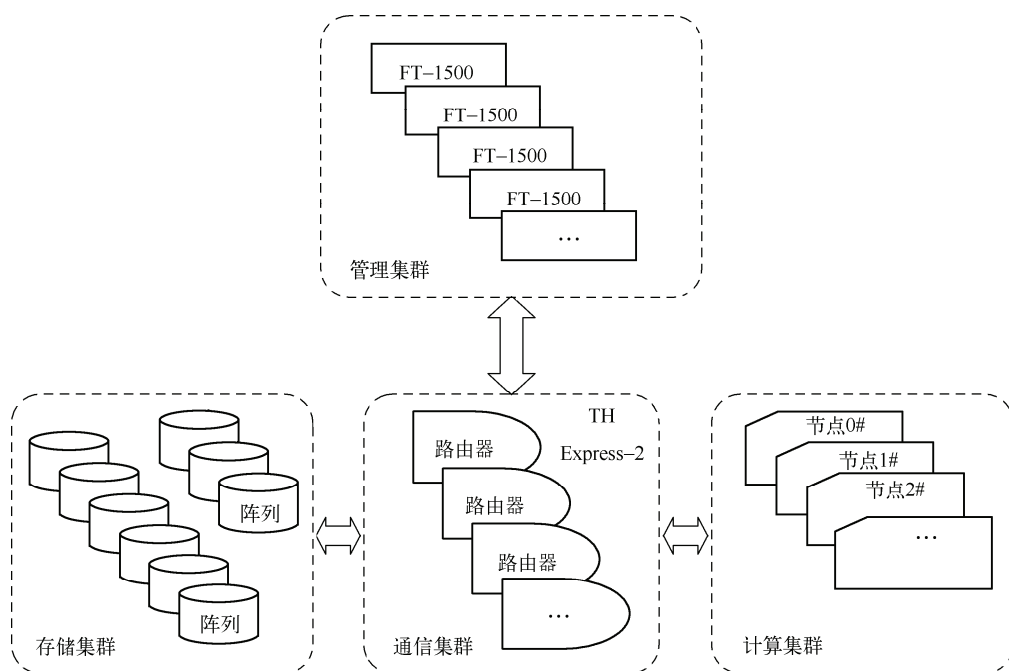


图 1-9 天河二号超级计算机的系统架构

用于前端处理器的管理集群共 8 个大型机柜，各节点共包含 4096 颗 FT-1500 处理器（国防科大研制，16 核 SPARC V9 架构，40 nm 制程，1.8 GHz 主频，峰值性能 144 GFLOPS）。管理集群的主要功能是管理整个计算机系统，包括故障诊断、大型计算任务在计算节点上的分配和调度，以及权限和通信资源的分配管理。

存储集群由 24 个机柜组成，采用磁盘阵列技术，其硬盘阵列的总容量为 12.4 PB。

计算集群是整个系统的核心，由 125 个机柜组成，每个机柜容纳 4 个机框，每个机框容纳 16 块主板，每个主板设置有 2 个计算节点，共计 16000 个计算节点，每个节点拥有独立的 64 GB 内存，节点使用中央处理器及协处理器的架构布局。单块主板上包含 APU 和 CPM 两个模块，APU 承载 5 块 Xeon Phi 31S1P(61 核，使用时屏蔽掉 4 核)协处理器，CPM 部分承载 1 块 Xeon Phi 31S1P +4 颗 Xeon E5-2692 v2 (12 核)，模块之间以 CPU 内部提供的 PCI-E 3.0 16X 接口进行连接，实际为 PCI-E 2.0 16X，单通道数据传输率为 10 Gbps。

天河二号内部计算节点的结构如图 1-10 所示。

如图所示，天河二号的每个计算节点主要由 2 个 Xeon E5-2692 主处理器和 3 个 Xeon Phi 31S1P 协处理器组成，两个主处理器之间通过 QPI 标准互连。每个主处理器因集成了内存控制器，故可连接 32G 容量的 DDR3 内存。第一主处理器与 PCH 芯片组之间仍然通过 DMI 标准互连，主处理器与协处理器之间通过 PCI-E 3.0 标准进行连接。

此外，第一主处理器通过网络终端接口（NIC）与其他计算节点互连。管理节点通过主板上的千兆以太网接口和 PCH 芯片组对各主处理器进行管理，同时通过 PCH 芯片组、可编程逻辑器件（CPLD）利用智能平台管理总线（IPMB）实现对各协处理器的管理。

2013 年 6 月，天河二号以其峰值计算速度 54902.4 TFLOPS（万亿次浮点运算）和持续速度 33862.7 TFLOP 超越美国的泰坦超级计算机（峰值 27112.5 TFLOPS，持续 17590.0 TFLOPS），成为当年世界上计算能力最快的超级计算机，并一举跃登 TOP 500 榜首。

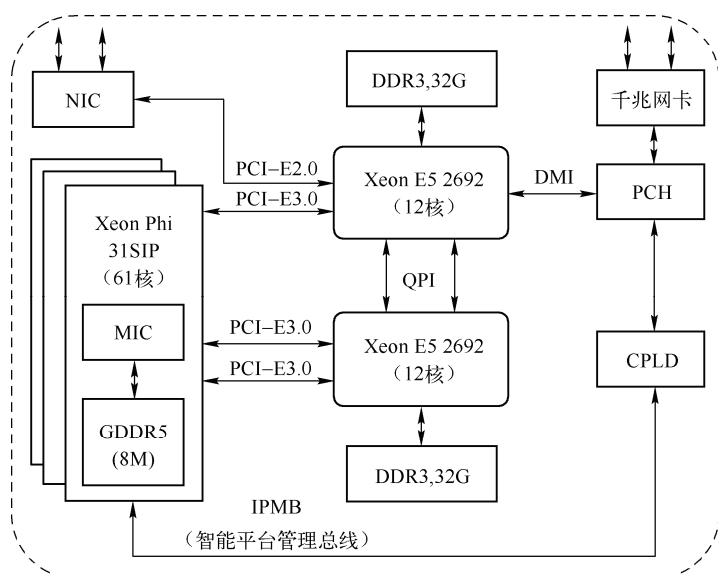


图 1-10 天河二号超级计算机的计算节点

根据天河二号的系统结构，可见整个系统由 15000 个计算节点组成，各节点之间通过网络连接。各计算节点具备一个完整计算机的功能，每个节点有自己的处理器和内存，且各节点共享同一个外存集群。此外，各节点是同级对等的，任何一个节点发生故障都不会影响到其他节点的正常运行，系统在管理集群的控制下，可以实现自动重构。

松散耦合型多处理机系统的特点是，用通信网络连接各节点，节点之间以中断方式传输信息包。每个节点内有一个 CPU、一个局部存储器，可能还有独自的外存或其他外设。通信网络的连接可以有星型、总线型、环型等多种模式。每个节点是一个计算机模块，模块内用局部总线连接了一个处理器 CPU、局部存储器 LM、一些 I/O 设备等。消息传送系统 MTS 将若干个这样的节点连接成多机系统。MTS 可以是比较简单的通信总线，也可以是比较复杂的互连网络，每个节点通过通信接口 CAS 与 MTS 相连。

从天河二号具备的这些特点来看，它具备松散耦合型多处理机系统的全部特征，因此天河二号超级计算机是一种典型的松散耦合型多处理机系统。

1.3.2 软件系统

计算机软件通常包含各类程序和文件。一般来讲，程序是用字符和符号描述的某种算法的实现过程，最终体现为机器指令序列，要被计算机硬件执行。文件则是对编制程序和运行、维护程序所作的说明，如对编程工具与运行环境的说明、帮助提示信息及其他参考信息等。在计算机系统中，各种软件有机地组合起来，构成了软件系统。从软件的功能和配置的角度出发，可将软件分为系统软件、应用软件两大类。

1. 系统软件

系统软件作为一种基础软件，其功能是负责系统的调度管理，提供程序的运行环境和开发环境，向用户提供各种服务，以保证计算机系统能够良好地运行。从配置的角度来看，系统软件是用户所使用的计算机系统的一部分，通常作为系统资源（软设备）提供给用户使用。常见的系统软件如下。

(1) 操作系统 (Operation System, OS)

操作系统是软件系统的核心,负责管理和控制计算机系统的硬件资源、软件资源和运行的程序。它是用户和计算机之间的接口,为用户提供了软件的开发环境和运行环境。

一个典型的操作系统由处理机调度、存储管理、设备管理、文件系统、作业调度等模块组成。这些模块相互配合,以尽可能优化的方式来调度管理系统的硬件、软件资源,合理地组织工作流程,提高系统的工作效率。例如,在具备多道程序运行环境的计算机系统中,需要处理机调度模块对处理机的分配和运行进行有效的管理;需要存储管理模块为各程序分配内存空间,提供内存保护,使这些程序互不干扰;需要设备管理模块为用户程序分配 I/O 设备,提供良好的人机界面,完成相关的 I/O 操作;需要文件系统模块为大量的、以文件形式组织和保存的信息提供管理;需要作业调度模块对以作业形式存放在外存中的用户程序进行调度管理,将它们从外存调入主存,交由 CPU 运行。所以,操作系统是对计算机系统的硬件、软件资源和运行的程序进行合理、高效的控制和管理。

对用户而言,往往通过操作系统提供的各种功能来操作和使用计算机的,如 DOS 操作系统提供的键盘命令、Windows 操作系统提供的人机界面等。人们通过按键或移动鼠标,可以打开、关闭计算机,可以使用计算机完成任意合法操作。操作系统提供了用户和计算机之间的接口。

当我们在开发用户程序时,常常需要调用在操作系统管理下的某些软件资源,以便得到操作系统的最大支持。所编制的程序一般也作为文件,由操作系统进行管理。程序运行时可能要调用操作系统管理的其他资源,使用操作系统的有关功能。所以,操作系统为用户提供了软件的开发环境和运行环境。

(2) 语言处理程序

用户通常使用程序设计语言来编写源程序,程序设计语言可以是 BASIC、Pascal、C、C++ 等高级编程语言,也可以是较低级的汇编语言。但是计算机硬件只能识别和执行由二进制代码表示的指令序列,所以需要语言处理程序将源程序转换为指令序列,即用机器语言表示的目标程序。有两种基本的转换方式,一种是解释方式,另一种是编译方式。相应地,语言处理程序就分为解释程序和编译程序两种。

在解释方式中,操作系统调用某种编程语言的解释程序(如 BASIC 语言的解释程序),计算机执行该解释程序,将源程序逐段地转换为具有相同功能的指令序列。转换完一段源程序,就执行该段对应的指令序列;再转换下一段源程序并执行下一段指令序列,直到整个源程序被解释执行完。这是一种边翻译、边执行的工作方式,目标代码的执行始终离不开源程序和解释程序。

在编译方式中,操作系统调用的是某种编程语言的编译程序(如 C 语言的编译程序),计算机执行该编译程序,将整个源程序全部转换为指令序列,即可执行的目标程序。然后不再需要源程序和编译程序,由计算机单独执行目标程序。这是多数程序设计语言采用的处理方式:先翻译、后执行。

将用汇编语言编写的源程序转换为目标程序,这一过程称为“汇编”,也属于编译类型。用于转换的程序称为汇编程序,它与解释程序、编译程序一样,都是语言处理程序。在剖析某些重要的软件时,常常需要将目标程序转换为用汇编语言表示的程序,这是汇编的逆过程,叫做“反汇编”。相应地,反汇编需要用反汇编程序来实现。

(3) 数据库管理系统

随着计算机技术在信息管理领域的广泛应用,数据管理的重要性更加突出,在这种背景下出现了数据库技术。所谓数据库,是指在计算机存储器中合理存放的、相互关联的数据集合,能提

供给不同的用户共享使用。在计算机中需配置数据库管理系统 (DBMS), 负责数据装配、内容更新、查询检索、通信控制, 以及对用数据库语言编写的程序进行翻译, 控制相关的运行操作等。如 SQL Server、Sybase、Oracle、Informix 等都是常见的数据库管理系统。

(4) 各种服务性支撑软件

为了帮助用户使用和维护计算机, 向用户提供服务性手段而编制的一类程序统称为服务性程序。这类程序一般是指输入和装配程序、编辑程序、调试程序、诊断程序、提示系统、窗口软件, 以及一些可供调用的通用性应用软件, 如文字处理软件、表格处理软件、图形处理软件等。这些服务程序往往作为操作系统可以调用的文件, 根据需要进行配置, 也可看成操作系统可以扩充的部分。

为了使用户能够更有效、更方便地操作计算机, 现在软件开发中的一个重要趋势是将开发及运行过程中所要用到的各种软件集成为一个综合的软件系统, 称为软件平台。这种软件平台技术为用户提供了一个完善的集成环境, 具有良好的人机界面和完善的服务支持。

(5) 各种标准程序库

这是由系统事先配置的通用、优化的标准子程序, 作为库文件可供用户调用。例如, 许多编译程序中含有库文件, 在对用户编写的源程序进行编译时, 编译程序根据源程序中给出的调用名, 便可调出相应的库文件进行装配。

2. 应用软件

应用软件直接面向用户需要, 是用户在各自的应用领域中, 为解决各类问题而编写的程序。计算机的应用领域极其广泛, 因而应用软件几乎涉及各行各业。按照不同的应用目的, 应用软件大致可划分为以下几种: 科学计算类程序、工程设计类程序、数据处理类程序、信息管理类程序、自动控制类程序和情报检索类程序。

尽管将计算机软件划分为系统软件和应用软件两大类, 但这种划分并不是一成不变的, 一些具有通用价值的应用软件也可以归入系统软件的范畴, 作为一种软件资源提供给用户使用。例如, 前面提到的数据库管理系统是面向信息管理应用领域的, 就其功能而言属于应用软件, 但在计算机系统中需要事先配置, 所以又是系统软件的一部分。

1.3.3 硬件、软件系统层次结构

计算机系统以硬件为基础, 通过配置各种软件来扩充系统功能, 形成一个有机组合的复杂系统。为了对计算机系统的有机组成建立整机概念, 便于对系统进行分析、设计和开发, 我们常常采用一种层次结构的观点, 将计算机系统从不同的角度分为若干层次。在分析计算机的工作原理时, 可以根据不同的工作需要, 从某一层去观察、分析计算机的组成、性能和工作机理。例如, 人们要了解一个计算机的硬件功能, 可以从该机器的指令系统级入手进行分析; 又如, 要了解主机对外设的控制情况, 可以从汇编语言级分析设备驱动程序。在设计或构造一个计算机系统时, 也常常分层次进行, 如在硬件的基础上逐级配置软件资源, 逐级扩展功能。在开发应用程序时, 也可按不同层次的模块进行, 如编程人员可以面向用户、面向系统分别编制不同功能的模块。

总之, 按分层结构化设计策略实现的系统不仅易于建造、调试和维护, 也易于扩充。根据不同的需要和目的, 有多种划分方法, 下面主要介绍两种常见的层次结构模型。

1. 从软件、硬件组成角度的层次结构模型

这种层次结构表明了一个计算机系统包括哪些硬件和软件, 并描述了硬件、软件之间的关系,

如图 1-11 所示。层次结构模型分为 8 层，其中微程序级和逻辑部件级属于硬件部分，传统机器级可看作硬件、软件之间的界面，其他都属于软件部分。从下层向上层发展，反映了计算机系统逐级生成的过程；从上层向下层观察，则可以帮助我们了解应用计算机求解问题的过程。

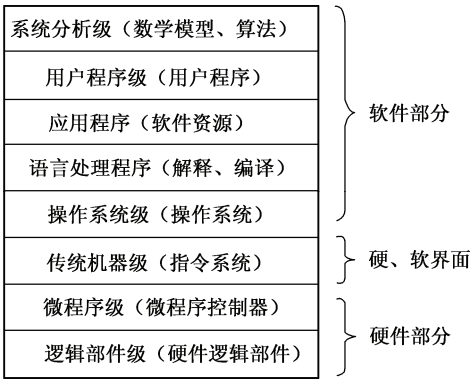


图 1-11 从软件、硬件组成角度的层次模型

（1）自下而上，计算机系统的逐级生成过程

① 拟定指令系统。先规定指令系统所包含的各种基本功能，这些功能都要由硬件来实现；各种软件最终也要转换为指令序列，才能被硬件识别和执行。所以，指令系统，即传统机器级，是连接硬件和软件的界面。指令系统常常用汇编语言来描述，便于分析和设计，但硬件最终执行的仍然是用机器语言表示的二进制代码。

② 创建硬件系统。根据指令系统来设计和实现硬件系统。硬件系统也称为硬核，其核心是 CPU 和主存，通过系统总线和接口将各种外存和外部设备连接起来，构成整机系统。从工作机制看，很多计算机采用微程序控制方式，即用微程序控制器来解释指令和执行指令，因而常常将硬件分为两级：下面一级是用连线连接的各种逻辑部件，称为硬连逻辑，包括寄存器和门电路；上面一级是微程序控制器，它通过执行微程序，发出各种微命令，控制各逻辑部件的正常工作。

③ 配置操作系统。操作系统是系统软件的核心和基础，因此在有了硬件系统之后，首先要配置操作系统，然后根据硬件系统的特点改进、扩展操作系统，不断推出新的版本。例如，在个人计算机上首先配置了单用户操作系统 PC-DOS，后来又推出了多任务操作系统 Windows。一些操作系统的核心部分源程序可以配置到多种计算机上，具有一定程度的通用性。一种计算机通常只需配置一种操作系统就可以工作，但为了适用于不同的应用领域，有些计算机也配置了多种操作系统。

④ 配置语言处理程序及各种软件资源。根据系统需要，配置相应的语言处理程序，如某种编程语言的编译程序、解释程序或汇编程序，并配置所需的各种软件资源。将这些软件归于操作系统的调度管理之下，形成一些通用的，或面向某种应用领域的软件平台，供用户随时调用。

⑤ 输入用户程序。当组成了一个硬件、软件配置完备的计算机系统之后，便可输入用户编写的应用程序，由计算机处理执行。

（2）自上而下，应用计算机求解问题的过程

① 系统分析级。系统分析人员根据对任务的需求分析，设计算法，构建数学模型，并根据数学模型和算法进行概要设计和详细设计。

② 用户程序级。编程人员根据详细设计，选择某种程序设计语言编写用户应用程序。

③ 操作系统级。计算机在操作系统的管理之下调用语言处理程序，如编译、解释或汇编程序，将用户源程序转换为用机器语言描述的目标程序。在源程序的输入、编辑、编译和调试过程中，通常要调用软件开发平台所提供的各种有关的软件资源。

④ 传统机器级。所形成的目标程序是用机器语言描述的指令序列，这些能够被计算机硬件识别和执行的二进制代码构成了可执行文件，即可执行的目标代码。我们从这一级看到的程序和计算机的工作属于传统机器级，或称为机器语言级。

⑤ 硬件系统级。由硬件执行机器语言程序，完成指令规定的操作。一般用户所看到的计算

机工作到这一级就可以了。但对于硬件设计者和维护人员来说，还需要了解硬核的工作情况，因此要深入到微程序级和逻辑部件级。对集成电路制造者而言，还要细化到电路一级有时甚至有可能到元器件一级。

2. 从语言功能角度划分的层次结构模型

图 1-12 是一个语言功能层次模型。这种层次划分的出发点是将计算机功能简化为执行由若干种语言编写的多个程序。

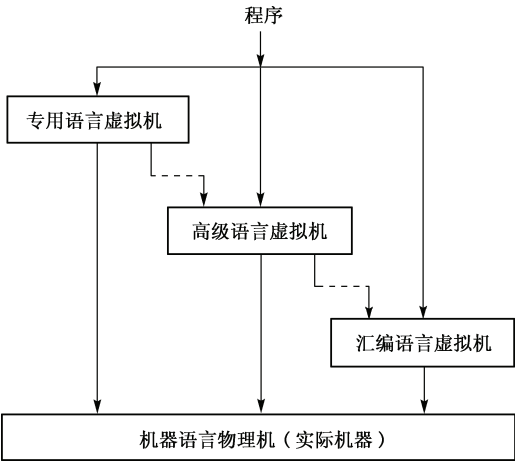


图 1-12 从语言功能角度的层次结构模型

程序，但实际的物理机只能执行机器语言指令，它通过配置汇编程序才能处理汇编语言程序。所以，用户在汇编语言这一级看到的是一台能执行汇编语言功能的虚拟机。

与算法、数学模型甚至自然语言接近的程序设计语言称为高级语言，如 Pascal、C、C++ 等通用的高级程序设计语言。用户在这一级看到的计算机是高级语言虚拟机。例如，配置了 C 编译程序的计算机对用户而言，能执行 C 语言程序，因而是一台具有 C 语言功能的虚拟机；配置了 BASIC 解释程序的计算机对用户而言，则是一台能执行 BASIC 语言程序的虚拟机。

对某些特殊的应用领域或者特定的用户，也可以使用某种专用语言。例如，在数字电路设计中，使用某种虚拟硬件描述语言 VHDL（Virtual Hardware Description Language）来描述电路中的组成与连接情况；在数据库应用中，使用某种数据库语言来创建和管理数据库等。用户在这一级看到的是具有某种专用语言功能的虚拟机。

在图 1-12 中，各层之间有什么关系呢？我们知道，用程序设计语言编写的程序必须翻译为机器语言程序，才能被计算机理解和执行。通常，大多数编程语言是直接翻译为机器语言的，如图中实线所示，汇编语言、高级语言、专用语言都可以直接翻译为机器语言。少数高级语言也可能采用间接翻译的方式，先转换为层次较低的中间语言，再由中间语言翻译为机器语言。如图中虚线所示，专用语言先翻译为高级语言，高级语言再翻译为机器语言；或者高级语言先翻译为汇编语言，汇编语言再翻译为机器语言。

采用虚拟机概念是计算机设计中的一个重要策略，它将计算机提供给用户的功能抽象出来，脱离具体的物理机器，使用户能够摆脱真实机器细节的约束。例如，许多系统软件的层次结构常分为虚拟层和物理层，在虚拟层上开发的系统软件具有较强的通用性，只要改变其与物理层的接口，就能应用在不同的物理机上。

1.3.4 硬件、软件功能划分与逻辑等价

计算机系统以硬件为基础,通过配置软件扩充其功能,并采用执行程序的方式来体现其功能。一般来说,硬件只完成最基本的功能,复杂的功能往往通过软件来实现。但是硬件与软件之间的功能分配关系常常随着技术发展而变化,哪些功能分配给硬件,哪些功能分配给软件是没有固定模式的。实际上,在计算机中,有许多功能既可以直接由硬件实现,也可以在硬件支持下依靠软件来实现,对用户而言在功能上是等价的,这种情况称为硬件、软件在功能上的逻辑等价。例如,乘法运算可以由硬件乘法器实现,也可以在加法器和移位器的支持下,通过执行乘法子程序实现。因而在用户看来,乘法器和乘法子程序在实现乘法运算功能上是没有区别的。那么,在设计一个计算机时,如何恰当地分配硬件、软件的功能?这既取决于所选定的设计目标、系统的性能价格比等因素,也与当时的技术水平有关。

早期曾采用“硬件软化”的技术策略。刚出现计算机时,各种基本功能均通过硬件来实现。随后为了降低造价,只让硬件完成较简单的指令操作,如传送、加法、减法、移位和基本逻辑运算,乘法、除法、浮点运算等较复杂的功能交给软件来实现。这导致了在当时条件下小型计算机的出现。“硬件软化”使小型机结构简单,又具有较强的功能,推动了计算机技术的普及与应用。

随着集成电路技术的飞速发展,人们可以将功能很强的模块集成在一块芯片上,于是出现了“软件硬化”的情况,将原来依靠软件才能实现的一些功能改由大规模或超大规模集成电路直接实现,如浮点运算、存储管理等。这就使系统具有更高的处理速度,在软件支持下有更强的功能。

微程序控制技术的出现使计算机结构和硬、软件功能分配发生了变化,对指令的解释与执行是通过运行微程序来实现的,因此出现了另一种技术策略“软件固化”。利用程序设计技术和扩大微程序的容量,可以使原来属于软件级的一些功能纳入微程序一级。微程序类似于软件,但被固化在只读存储器中,属于硬件 CPU 的范畴,称为固件。这种方式使 CPU 的结构得到简化。另外,人们也常采用软件固化的策略,将系统软件的核心部分(如操作系统的内核、常用软件中固定不变的部分)固化在存储芯片中。从用户的角度看,它们属于系统硬件(如系统板)的一部分。例如,IBM-PC 微机系统将操作系统中的基本输入/输出系统 BIOS 固化在系统板上,Pentium 微处理器将存储管理功能集成于 CPU 芯片之内,等等。

系统设计者必须关心硬件、软件之间的界面如何划分,决定系统功能哪些由硬件实现,哪些由软件实现。用户则更关心系统究竟能提供哪些功能,至于这些功能是由硬件实现还是由软件实现,在逻辑功能上是等价的,只是执行速度不同而已。

尽管我们常按层次结构来分析、设计计算机系统,但考虑到初学者的认识过程,本书不准备机械地分级讨论计算机组成原理,而是以讨论计算机各组成部件原理及整机连接为基线,即分别讨论 CPU 组织、存储器组织、外围设备组织、以总线和接口为基础的 I/O 系统组织。但在分析这些计算机组织时,力求注意它们的硬件、软件功能分配和界面关系,抽象出相应的概念模型,从寄存器级、微操作控制级、机器指令级等几个层次去观察分析计算机的硬件组成和工作机制。例如,分析 CPU、存储器、接口、I/O 设备中有哪些寄存器,以及与寄存器相当的部件,如加法器、输入/输出通道、选择器等,了解它们通过什么样的数据通路结构相连接。从寄存器级进行描述容易获得清晰的结构概念。

CPU 的功能特性体现为指令系统,而硬件的任务就是执行指令,实现寄存器级的信息传送。为此,可先将指令流程分解为若干步寄存器级传送操作,再从微操作控制这一级产生微操作命令序列,控制完成寄存器级的传送操作。微操作命令的产生既可以通过组合逻辑控制方式实现,也可以通过微程序控制方式实现。

1.4 计算机的特点与性能指标

1.4.1 计算机的特点

由于数字计算机采用数字化的信息表示方法和存储程序工作方式，能自动地对各种数字化信息进行算术和逻辑运算，并能进行广泛的信息处理。其主要特点表现在以下几方面。

① 能自动地执行程序。输入程序后，只要符合运行条件，计算机便能自动地从起始地址读取程序，解释并执行程序。这也是计算机区别于其他计算工具的本质特点。

② 运算速度快。计算机硬件目前是由高速的电子线路组成的，工作速度极快。一个复杂的数学计算或大量的数据处理可能需要若干个人工作很长一段时间才能完成，用计算机来处理很快就可以获得结果。这不仅极大地提高了人类的工作效率，也大大增强了人类处理问题的能力，使许多复杂问题能实际得到解决。随着更高速的新器件的诞生，以及系统结构的进一步优化，计算机的运算速度还将得到更大的提高。

③ 计算精度高。计算机采用数字代码表示数据，代码的位数越多，数据的表示精度就可以越高。尽管在实际计算机中考虑到硬件成本等因素，对数据的基本位数有一定限制，但通过软件可以实现多位数据的运算，能获得更高的计算精度。

④ 存储能力强。计算机依靠存储器能够存储大量的程序和数据，这是保证计算机能自动连续工作的先决条件。程序和数据是由二进制代码组成的，只要是具有两种稳定状态的物理介质都能存储二进制代码。在计算机中，主存利用触发器或电容电荷存储信息，外存利用磁化状态或其他介质状态存储信息。因而，计算机具有很强的信息存储能力，存储的程序和数据越多，计算机的处理功能也就越强。

⑤ 通用性好。由于各类信息都可以表示为数字化信息，都能被计算机处理，所以计算机的应用领域极其广泛。又由于计算机用数字逻辑部件作为处理数字信号的统一逻辑基础，因此计算机既能实现算术运算，又能实现逻辑运算；既能进行数值计算，又能对各类非数值信息进行处理，如信息检索、图像处理、语音处理、逻辑判断等。计算机的这种极好的通用性使它能应用到各行各业，并渗透到人们的工作、学习、生活等方面。

1.4.2 计算机的主要性能指标

在实际应用中，我们可能需要根据计算机的性能指标进行系统的软硬件配置或者评价某个计算机的优劣。常用来评价计算机综合性能的指标主要包括了 CPU 的运算速度、存储容量、数据的输入/输出能力等。

1. 基本字长

在计算机中，算术运算可以分为定点运算和浮点运算两大类。基本字长一般是指处理器中参加一次定点运算的操作数的位数，如 8 位、16 位、32 位或 64 位。基本字长影响着计算的精度、硬件的成本，甚至对指令系统功能也有影响。

在一次运算过程中，操作数和运算结果通过数据总线，在寄存器和运算部件之间传送。因此基本字长标志着计算精度，也反映了寄存器、运算部件和数据总线的位数。基本字长越长，操作数的位数越多，计算精度也就越高；但相应部件的位数也会增多，使硬件成本随之增加。另外，某些信息（如字符类信息）只需要用 8 位二进制代码来表示，8 位称为 1 字节。因此，为了较好地协调计算精度与硬件成本的制约关系，针对不同需求，大多数计算机允许采用变字长运算，即

允许硬件实现以字节为单位的运算以及某种基本字长（如 16 位）运算或双字长（如 32 位）运算，通过软件来实现多字长运算。

指令字长与数据字长之间也有一定程度的对应关系。基本字长较长的计算机，其指令的位数可能也较多，读取指令的速度和处理指令的效率要高些，指令系统的功能相应比较强，这在传统的小型机中表现较为明显。

2. 外频

外频也叫外部频率或基频，有时也称为系统时钟频率，是指主板上的振荡器输出的时钟频率，也是计算机中一切硬件部件工作所依据的基准时钟信号，它经过倍频系数放大后用作计算机中各部件的工作频率，如 CPU、内存和各类总线等。标准外频有 100 MHz、133 MHz，甚至 166 MHz、200 MHz，一般很少超过 300 MHz。

在计算机系统中，一般不会用外频来评价计算机的性能，但外频是其他一些频率指标如 CPU 主频、总线频率或者内存频率等的基础。

3. CPU 的运算速度

CPU 的运算速度是计算机的一项重要性能指标，一代又一代计算机追求的目标之一就是如何提高运算速度。由于计算机执行不同的运算所花费的时间可能会不同，如定点运算所需的时间较少，而浮点运算所用的时间较多，运算速度取决于诸多因素。

（1）CPU 的主频

CPU 的主频是指 CPU 内核的工作频率，通常所说的某款 CPU 是多少 GHz，就是指 CPU 主频，有时也叫 CPU 的时钟频率。CPU 主频=外频×倍频系数，提高任何一项都可以使 CPU 的主频上升（超频）。

CPU 主频的高低是决定计算机工作速度的重要因素，但两者之间并没有正比关系。在 CPU 时钟频率中，相邻两个时钟脉冲之间的间隔即一个时钟周期，它与 CPU 完成一步操作所需的时间是相对应的。例如，Intel 8088 的时钟频率为 4.77 MHz，80386 的时钟频率达到 33 MHz，80486 的时钟频率高达 100 MHz，Pentium 系列的时钟频率高达 300 MHz、500 MHz、1 GHz 甚至更高。如果了解某种运算所需的具体时间，则可根据该运算占用的时钟周期数，计算出所需时间。

（2）平均每秒执行的指令数 IPS

CPU 平均每秒执行指令的数量，即 IPS（Instructions Per Second），也用 MIPS 或 GIPS 来表示，这个指标适合用于评价标量运算，不适合评价向量运算。

虽然计算机的指令类型很多，各指令执行的时间和出现的频度都不会完全相同，但计算机在运行过程中所执行的大部分指令都是简单指令，因此用每秒平均执行的指令条数作为 CPU 的速度指标，在一定程度上反映出计算机的运算速度。特别是 RISC 型的 CPU，其所有指令几乎全是简单指令，更适合于用 IPS 来衡量其速度。例如，80486 的运算速度达到 20 MIPS，Pentium 超过 100 MIPS，ALPHA（RISC 微处理器）则高达 400 MIPS。

（3）平均每条指令的时钟周期数 CPI

CPU 平均每条指令的时钟周期数，即 CPI（Clock cycles Per Instruction），也常用来衡量 CPU 的运算速度。

CPI 是一个平均概念，其物理意义可以理解为：CPU 执行一个程序所需的时钟周期数与这个程序对应的指令数的比值，即 $CPI = m_c / n_i$ ，这里的 m_c 是这个程序的时钟周期数， n_i 是这个程序的指令数。

(4) 每秒执行定点/浮点运算的次数

每秒钟能够执行定点和浮点运算的次数也可以用来刻画计算机的运算速度。早期的计算机中常用定点次数来表示速度，如 DJS-1 计算机其运算速度为平均可执行 1800 次 32 位定点数运算。

对现代的高速计算机，因其主要进行浮点向量运算，故一般用单位时间内执行浮点运算的次数（Floating-point Operations Per Second, FLOPS）来表示速度。例如，国防科大研制出的“天河二号”超级计算机，其运算速度就表示为 33.86 PFLOPS。

4. 数据通路宽度与数据传输率

这两个指标主要是用来衡量计算机部件的数据传输能力。

(1) 数据通路宽度

数据通路宽度是指数据总线一次能并行传送的数据位数，也会直接影响计算机的处理性能。

数据通路宽度一般可分为 CPU 内部和 CPU 外部两种情况。CPU 的内部数据通路宽度一般与 CPU 的基本字长相同，也等于 CPU 内总线的位宽，CPU 的外部数据通路宽度则等于系统数据总线的位宽。有的 CPU 内外的数据通路宽度一样，有的则不同。例如，Intel 8088 处理器的 CPU 内部数据宽度是 16 位，外部数据宽度却只是 8 位，Intel 80386/80486 处理器的 CPU 内外数据宽度相同，两者都是 32 位。

(2) 数据传输率（Data Transfer Rate, DTR）

数据传输率也叫比特率，是指单位时间内信道的数据传输量，基本单位是 bps。在计算机或网络科学中，常常借用带宽（Bandwidth）一词来表示数据传输率，显然这里的带宽已与其原始含义不同，已发生了转义变化。数据传输率与传输信道的数据通路宽度和最大的工作频率有关，其通常的简化计算规则如式（1-1）所示：

$$DTR = \frac{D}{T} = W \times f \quad (\text{bps}) \quad (1-1)$$

其中，DTR 表示数据传输率， D 是数据的传输量， T 是相应的数据传输时间， W 是数据通路的宽度， f 是工作频率。例如，若 PCI 总线的位宽是 32 位，总线频率为 33.33 MHz，则总线的数据传输率（总线带宽）约为 133 MBps（忽略编码方式等，详情见式（5-1））。

实际上，对计算机而言，所有的硬件部件都会涉及位宽和带宽的概念。比如，Intel 平台上的 FSB（Front Side Bus，即前端总线）、AMD 平台的 HT（Hyper Transport）总线，甚至内存、网络设备等，都存在带宽的概念。不论对何种硬件部件，在理解其带宽概念时，只需紧紧把握住单位时间内传输的数据量这一特性即可。

5. 存储容量

存储容量用来衡量计算机的信息存储能力。计算机的存储器分为内存（主存）和外存（辅存）两类，因而存储容量相应地有内存容量和外存容量这两个指标。

(1) 主存容量

主存储器用来存放 CPU 当前需要执行的程序和需要处理的数据。主存容量越大，存放的信息也越多，就不会频繁地将数据从辅存（如硬盘，速度较慢）读入主存，这样减小了 CPU 等待数据读取的时间，从而提高了 CPU 的执行效率。因此，主存容量的大小，在一定程度上决定了计算机的综合性能。

主存容量就是指主存能存储的数据量，取决于主存的编址单元数和每个编址单元的位数（宽度）。CPU 根据主存地址码可以直接访问主存单元，有的主存以字节为编址单位，而有的主存则

以字长为编址单位，因此常用两种方式来表示主存容量：

① 字节数

微型计算机的主存多按字节编址，故每个编址单元有 8 位，所以可用字节数来表示主存容量的大小。例如，PC/XT 的主存容量为 1 兆字节，记为 1 MB (B 是字节 Byte 的缩写， $1\text{M}=2^{20}$ ， $1\text{K}=2^{10}$)；486 机或 Pentium 机的主存容量可达到 4 GB ($1\text{G}=2^{30}$)。

② 字数×位数

某些计算机的主存是按字编址的，每个编址单元存放一个字长的数据，字长等于 CPU 的基本字长，所以用字数（或单元数）×位数来表示主存容量，即表明主存包含多少个字单元，每个单元的宽度多少位。例如，某主存有 64×1024 个字单元，每个单元 16 位，则该主存的容量可表示为 $64\text{K}\times 16\text{bit}=128\text{KB}$ 。

主存的最大可编址单元数（编址空间）是由地址线的位数决定的。若地址线有 16 位，则主存的最大可编址单元数就只能是 $2^{16}=64\text{K}$ ；若地址线有 20 位或 32 位，则主存的最大可编址空间就是 1M 或 4G。若一个主存的实际可编址空间大于由地址线决定的最大可编址空间，则超出部分的可编址单元无法分配到地址码，因此这部分存储单元会因悬空而无法被系统使用。掌握地址线的位数与主存编址空间的对应关系，有利于存储器的逻辑设计。

(2) 辅存容量

辅存，也叫外存，指计算机系统中能够联机读写的外部存储器，如硬盘光盘等。因此，辅存容量就是指外部存储器能存储的最大数据量，一般也用 B、MB、GB 或 TB 等来表示。就目前主流的存储设备容量而言，B 和 MB 这两个单位显得太小，已很少直接使用它们了。例如，现在主流硬盘的容量一般为 250 GB、500 GB 或 1TB 等；其他外部存储器如 U 盘和光盘等，通常也是用 GB 或 TB 来表示。

计算机中的软件资源，如操作系统、编译程序以及应用程序等，除了正在执行的那部分，其余都是存放在外存中的，需要时再读入主存。外存不采用内存那样的编址方式，操作系统一般通过虚拟存储技术对它进行管理，因此其容量限制与地址总线位数无直接关系。

6. 外围设备配置

一台计算机所配置的外围设备种类，配置每种外围设备的数量或者容量等等，这些因素也都会影响整个计算机系统的性能。一般来说，外围设备配置越高级，计算机的整体性能越好。具体如何配置外围设备，这与用户的实际需要和经济支付能力有关。

通常情况下，在计算机系统的技术说明资料中常常给出最低配置规格、标准配置规格或者豪华配置规格等方案。比如，现在主流的显示器配置为分辨宽屏 LCD 液晶显示器，如果打游戏的话则需要配置游戏操纵手柄等。

7. 软件配置

软件配置包括安装什么版本的操作系统、编程语言和工具软件，以及其他应用程序的安装配置情况，这里也有允许配置和实际配置的问题。

通常情况下，软件配置得越合理，计算机的综合性能也越高。原则上，可以不断扩充软件配置，但实际配置软件时还应考虑所需要的硬件支持，如安装并运行某个操作系统需要多大的存储容量等问题。

习 题 1

1. 简要解释下列名词术语：

数字计算机 硬件 软件 CPU 主存储器 外存储器 外部设备
信息的数字化表示 存储程序工作方式 模拟信号 数字信号 脉冲信号
电平信号 系统软件 应用软件 操作系统 语言处理程序 物理机
虚拟机 总线 数据通路宽度 数据传输速率 接口 通道 字节 字长

2. 数字计算机的主要特点是什么？
3. 计算机有哪些主要性能指标？
4. 冯·诺依曼思想包含哪些要点？
5. 信息的数字化表示包含哪两层含义？
6. 用数字信号表示代码有什么优点？
7. 编译方式和解释方式对源程序的处理有什么区别？
8. 为什么要对计算机系统进行层次划分？
9. 软件系统一般包含哪些部分？试列出你所熟悉的几种系统软件。
10. 以你所熟悉的一种计算机系统为例，列举出该系统所用的 CPU 型号、时钟频率、字长、主存容量、外存容量、所连接的 I/O 设备的名称等。
11. 什么是控制流驱动？什么是数据流驱动？
12. 你是否曾在计算机的机器指令级、操作系统级、汇编语言级或高级语言级上做过工作或练习？或调用过该级的功能？举出所做的工作或所调用的功能名。
13. 试分析微型、小型和大型计算机的特点。

第2章 数据的表示、运算与校验

计算机是处理信息的工具，要了解计算机的工作原理，首先需要了解计算机中信息的表示方法。计算机中的信息可分为两大类。一类是计算机处理的对象，泛称为数据，又进一步分为数值型数据和非数值型数据（如字符、图像等）。另一类是控制计算机工作的信息，泛称为控制信息，计算机中的指令信息则是产生各种控制命令的基本依据。相应地，在计算机工作时将存在两类信息流：数据流和控制流。

本章将介绍数值型数据的表示（原码、反码、补码和移码，定点数和浮点数）、字符型数据的表示、数据的运算方法和常见的数据校验规则等。

2.1 数值型数据的表示

一个数值型数据的完整表示包含三方面：① 采用什么进位计数制，通俗地讲，就是逢几进位；② 如何表示一个带符号的数，即如何使符号数字化，这就涉及机器数的编码方法，常用的有原码和补码；③ 小数点如何处理，有两种方法，即定点和浮点表示。

2.1.1 进位计数制

通常，用若干数位的组合去表示一个数，形成一串代码序列（如 $X_n X_{n-1} \cdots X_0$ ）。如果从 0 开始计数以得到各种数值，就存在一个由低位向高位进位的问题，这种按一定进位方式计数的数制叫做进位计数制，简称进位制。我们也可以将它展开为多项式，这种形式能清晰地表明各数位之间的关系，可由此寻找各种进位制之间的转换规律。这种多项式的通式可写为：

$$\begin{aligned}(S)_r &= X_n R^n + X_{n-1} R^{n-1} + \cdots + X_0 R^0 + X_{-1} R^{-1} + X_{-2} R^{-2} + \cdots + X_{-m} R^{-m} \\ &= \sum_{i=-m}^n X_i R^i \quad (X_i \text{ 为 } 0、1、\cdots、r-1 \text{ 中的一个})\end{aligned} \quad (2-1)$$

式（2-1）中包含了 $n+1$ 位整数及 m 位小数。

数的进位制涉及两个基本概念：基数 r 和各数位的权值 r^i 。它们是构成某种进位制的两个基本要素。基数 r 是某种进位制中会产生进位的数值，等于每个数位中所允许的最大数码值（即 $r-1$ ）加 1，也就是各数位中允许选用的数码个数。例如十进制，每个数位允许选用 0~9 这 10 个数码中的某一个，基数 $r=10$ 。或者说，各数位中允许选用的最大数码值为 9，再加 1 就会逢 10 进位，因此基数为 10。

一个数码处在不同的数位上时它所代表的数值不同。例如在十进制中，个位的 1 表示 10^0 ，而百位上的 1 则表示 10^2 。因此在进位制中每个数位都有自己的权值，它是一个与所在数位相关的常数，称为该位的位权，简称为权。该位数码所表示的数值等于该数码本身的值乘以该位的权值。显然，各数位的权值是不同的。例如十进制数，从小数点往左，整数部分的位权依次是 10^0 ， 10^1 ， 10^2 ，…。从小数点往右，小数部分的位权依次是 10^{-1} ， 10^{-2} ，…。在十进制数 139 中，十位上的数值等于 3×10^1 。不难看出，相邻两位的权值之比恰好等于基数 r ，如 $10^2/10^1=10$ 。

1. 计算机中常用的进位制

在日常生活和数学运算中，我们一般采用十进制数的形式表示数值，因此在编程中也采用十

进制数的形式表示数值型数据。但计算机硬件采用二进制数的形式，即用 0、1 代码表示数字。这就需要进行十进制数与二进制数之间的转换。为了与日常使用的十进制数相衔接，计算机中也可以部分采用“二-十”进制，即用 4 位二进制数来表示 1 位十进制数。程序与数据需存入计算机的存储器中，存储器分为许多存储单元（编址单元），为各单元分配地址码，编程时常用八进制数或十六进制数表示地址码。实际上，八进制数和十六进制数都可以视为二进制数的一种缩写形式。本节将分别说明几种进位制的表示方法，然后讨论它们之间的相互关系和转换方法。

(1) 二进制

在二进制中，每个数位仅能选择 0、1 这两个数码中的一个（非 0 即 1），逢 2 进位或借 1 当 2，基数 $r=2$ 。

若采用代码序列形式，其通式可写为 $(X_n X_{n-1} \cdots X_0 . X_{-1} X_{-2} \cdots X_{-m})_2$ 。代码序列本身只给出了各数位的数码，而基数及各位的权值都是隐含约定的，为了区别于其他进位制，可给代码序列加上括号，在括号下标处注明基数值。

若采用多项式形式，则基数值与各数位的权值均包含在多项式中。可由式 (2-1) 导出二进制数的通式如下：

$$\begin{aligned}(S)_2 &= X_n 2^n + X_{n-1} 2^{n-1} + \cdots + X_0 2^0 + X_{-1} 2^{-1} + X_{-2} 2^{-2} + \cdots + X_{-m} 2^{-m} \\ &= \sum_{i=-m}^n X_i 2^i \quad (x_i \text{ 为 } 0 \text{ 或 } 1)\end{aligned}\quad (2-2)$$

式 (2-2) 表明基数为 2， $(S)_2$ 表示一个二进制数 S 。整数共 $n+1$ 位，对应于代码序列 $X_n X_{n-1} \cdots X_0$ ， X_0 的位权是 2^0 ， X_n 的位权是 2^n 。小数共 m 位，对应于代码序列 $X_{-1} X_{-2} \cdots X_{-m}$ ， X_{-1} 的位权是 2^{-1} ， X_{-m} 的位权是 2^{-m} 。

【例 2-1】 $(101.01)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = (5.25)_{10}$ 。

(2) 八进制

在八进制中，每位可选用的数码有 8 个：0~7。逢 8 进位，基数 $r=8$ 。

八进制数的多项式形式为：

$$(S)_8 = X_n 8^n + X_{n-1} 8^{n-1} + \cdots + X_0 8^0 + X_{-1} 8^{-1} + X_{-2} 8^{-2} + \cdots + X_{-m} 8^{-m} \quad (2-3)$$

式 (2-3) 表明了八进制数的基数值 $r=8$ ，各位数的位权为 8^i 。

【例 2-2】 $(703.64)_8 = 7 \times 8^2 + 0 \times 8^1 + 3 \times 8^0 + 6 \times 8^{-1} + 4 \times 8^{-2} = (451.8125)_{10}$ 。

在例 2-2 中，703.64 是一个八进制数的代码序列，例中给出了它的多项式，451.8125 是转换为十进制数后的代码序列。

八进制数与二进制数之间可以分段对应转换，即用 3 位二进制数对应 1 位八进制数，因而又称为“二-八”缩写。一位八进制数可表示的最大值是 7，而 3 位二进制数可表示的最大值也是 7。当由二进制数转换为八进制数时，整数部分自小数点向左、小数部分自小数点向右，每 3 位一段，然后分段转换为对应的八进制数。反之，当由八进制数转换为二进制数时，将每 1 位八进制数转换为 3 位二进制数即可。

【例 2-3】 $(703.64)_8 = (111000011.110100)_2$ 。

(3) 十六进制

在十六进制中，每位可选用的数码共 16 个，相当于十进制中的 0~15，书写为 0、1、…、8、9、A、B、C、D、E、F，逢 16 进位，基数 $r=16$ 。

十六进制数的多项式形式为：

$$(S)_{16} = X_n 16^n + X_{n-1} 16^{n-1} + \cdots + X_0 16^0 + X_{-1} 16^{-1} + X_{-2} 16^{-2} + \cdots + X_{-m} 16^{-m} \quad (2-4)$$

式(2-4)表明了十六进制数的基数值为16,各位的位权为 16^i 。

【例 2-4】 $(BC3.89)_{16}=11\times 16^2+12\times 16^1+3\times 16^0+8\times 16^{-1}+9\times 16^{-2}\approx (3011.535)_{10}$ 。

在例 2-4 中, BC3.89 是一个十六进制数的代码序列,例中给出了它的多项式,3011.535 是转换为十进制数后的代码序列。此外,十六进制数的标注方法还有一种,即以 H 为后缀,如 BC3.89H。

与八进制数类似,十六进制数与二进制数之间也可以分段对应转换,即用 4 位二进制数对应 1 位十六进制数,因而又称为“二-十六”缩写。1 位十六进制数可表示的最大值是 15,而 4 位二进制数可表示的最大值也是 15。当由二进制数转换为十六进制数时,整数部分自小数点向左、小数部分自小数点向右,每 4 位一段,然后分段转换为对应的十六进制数。反之,当由十六进制数转换为二进制数时,将每 1 位十六进制数转换为 4 位二进制数即可。

【例 2-5】 $(BC3.89)_{16}=(101111000011.10001001)_2$ 。

(4) 二-十进制

日常生活中人们习惯使用十进制数,但计算机内部却以二进制数表示为基础,那么计算机如何对十进制数进行运算处理呢?有两种方法可供选择。

一种方法是在编制程序时使用十进制数,输入计算机后,用“十翻二”程序将运算结果转换为二进制数处理,再用“二翻十”程序将运算结果转换为十进制数,以供输出。如果计算机所需处理的是科学计算一类的任务,原始数据量不大而运算处理比较复杂,则可以采取这种方式。

另一种方法是采用“二-十”进制表示,即用 4 位二进制数表示 1 位十进制数。这种用二进制编码表示十进制数的编码方法称为 BCD(Binary Coded Decimal)码。那么,1 位十进制数又用什么样的二进制编码来表示呢?最常用的方法是取常规二进制中 0~9 这 10 种编码去对应十进制数 0~9。从高位起,这 4 位的位权依次为 2^3 、 2^2 、 2^1 、 2^0 ,即 8、4、2、1,所以这种二-十进制编码又称为“8421 码”。从每位二进制码看,基数为 2。但从二-十进制码(即 4 位二进制码)看,与常规十进制数相同,只允许选用 0~9 中的一个,逢 10 进位,基数为 10。如果计算机处理的是数据统计(如财务管理)类任务,原始数据量大而计算处理比较简单,可以在编程时采用十进制数而在计算机内部采用“二-十”进制数。十进制数与二-十进制数之间可以分段对应,可用简单的硬件电路实现转换。

【例 2-6】 $(137)_{10}=(000100110111)_{BCD}$ 。

当由十进制数转换为二-十进制数时,自高位起,将各位十进制码分别用 4 位二进制码代换。由二-十进制数转换为十进制数时,从高位或从低位起每 4 位一组,分别将各组的 4 位二进制码直接转换为十进制码。

表 2-1 给出了 4 位二进制数与其他进位制数之间的对应关系。如前所述,在编写程序时,通常用十进制数表示数值,用十六进制数或八进制数表示地址码。在计算机内部硬件操作时,用二进制数或二-十进制数表示数值,用二进制表示地址码。

表 2-1 常用进位制之间的对应关系

| 十进制数 | 二进制数 | 八进制数 | 十六进制数 | 二-十进制数 |
|------|------|------|-------|--------|
| 0 | 0000 | 0 | 0 | 0000 |
| 1 | 0001 | 1 | 1 | 0001 |
| 2 | 0010 | 2 | 2 | 0010 |
| 3 | 0011 | 3 | 3 | 0011 |
| 4 | 0100 | 4 | 4 | 0100 |
| 5 | 0101 | 5 | 5 | 0101 |
| 6 | 0110 | 6 | 6 | 0110 |

续表

| 十进制数 | 二进制数 | 八进制数 | 十六进制数 | 二 - 十进制数 |
|------|------|------|-------|----------|
| 7 | 0111 | 7 | 7 | 0111 |
| 8 | 1000 | 10 | 8 | 1000 |
| 9 | 1001 | 11 | 9 | 1001 |
| 10 | 1010 | 12 | A | 00010000 |
| 11 | 1011 | 13 | B | 00010001 |
| 12 | 1100 | 14 | C | 00010010 |
| 13 | 1101 | 15 | D | 00010011 |
| 14 | 1110 | 16 | E | 00010100 |
| 15 | 1111 | 17 | F | 00010101 |

2. 各种进位制之间的相互转换

上述进位制可以分为两组。一组是二进制、八进制、十六进制，它们之间可以分段对应转换，比较简单。另一组是十进制、二 - 十进制，它们之间也可以分段对应转换。两组之间则需通过二进制与十进制的转换来实现。例如，将一个十六进制数转换为二 - 十进制数，可先将这个十六进制数转换为二进制数，再将二进制数转换为十进制数，最后转换成二 - 十进制数。所以，进位制之间的转换重点是要解决二进制与十进制之间的转换问题。

二进制数与十进制数之间不存在简单的分段对应关系，需要分成整数和小数两部分，分别按相应转换算法进行整体转换。由于多项式形式比代码序列形式的数学关系更明确，所以以式 (2-2) 为基础去寻找转换规律。基本思路是按位权关系逐位分离（由十进制数转换为二进制数），或各位相加（由二进制数转换为十进制数）。

(1) 十进制整数转换为二进制整数

为了实现按权逐位分离，通常有下述两种方法可供选择。

① 减权定位法

若转换后的二进制数代码序列为 $X_nX_{n-1}\cdots X_1X_0$ ，则从高位起将十进制数依次与二进制数各位的权值进行比较。若够减，则对应位 $X_i=1$ ，减去该位权值后继续往下比较；若不够减，则对应位 $X_i=0$ ，越过该位后继续往下比较；如此反复进行，直到所有二进制位的位权都比较完毕为止。

【例 2-7】 将 $(116)_{10}$ 转换为二进制数。

因为 $128 > 116 > 64$ ，所以从权值 64 开始比较。

| 减权比较 | X_i | 位权 |
|-----------------|-------|----|
| $116 - 64 = 52$ | 1 | 64 |
| $52 - 32 = 20$ | 1 | 32 |
| $20 - 16 = 4$ | 1 | 16 |
| $4 < 8$ | 0 | 8 |
| $4 - 4 = 0$ | 1 | 4 |
| $0 < 2$ | 0 | 2 |
| $0 < 1$ | 0 | 1 |

转换结果： $(116)_{10} = (1110100)_2$ 。

注意两点：第一步求得的是最高位；差值为 0 时不一定就是比较结束，必须对全部二进制数位都比较完毕（即比较到位权为 1）。

减权定位法的思路是：从高位起，由十进制数中逐位分离出二进制数位。这种方法比较直观，适于手算；但各步骤操作不统一，程序实现较烦琐。

② 除基取余法

分析二进制数的多项式形式会发现：将式 (2-2) 的整数部分除以二进制的基数 2，得到下式

$$\frac{S}{2} = (X_n 2^{n-1} + X_{n-1} 2^{n-2} + \cdots + X_1 2^0) + \frac{X_0}{2} \quad (2-5)$$

显然，式(2-5)的括号之内是除2操作后得到的商，而 X_0 是余数。若余数为0，表明 $X_0=0$ ；若余数为1，则表明 $X_0=1$ 。继续对商除以基数2，可依次判定多项式的各项系数（即二进制代码序列的各位） $X_1 \sim X_n$ 。

由此得到除基取余法的转换方法：将十进制整数除以2，所得余数作为对应的二进制数低位的值；继续对商除以2，所得的各次余数就是二进制数的各位值；如此进行，直到商等于0为止。注意，最后一项余数为二进制数最高位的值。

【例 2-8】 将 $(116)_{10}$ 转换为二进制数。

| | 余数 | 二进制数位 |
|---------------------|-------|---------|
| 2 $\overline{)116}$ | | |
| 2 $\overline{)58}$ | 0 | $X_0=0$ |
| 2 $\overline{)29}$ | 0 | $X_1=0$ |
| 2 $\overline{)14}$ | 1 | $X_2=1$ |
| 2 $\overline{)7}$ | 0 | $X_3=0$ |
| 2 $\overline{)3}$ | 1 | $X_4=1$ |
| 2 $\overline{)1}$ | 1 | $X_5=1$ |
| 0 | 1 | $X_6=1$ |

转换结果： $(116)_{10} = (1110100)_2$ 。

除基取余法也是一种逐位分离法。与减权定位法不同，它从整数部分的低位开始分离，通过除以基数实现。由于每一步都是重复执行除2运算，各步操作统一，适于编程实现。当然，由于计算机中并无真正的十进制数，只能用二-十进制数来代替十进制数，编程实现的实际上是二-十进制数到二进制数的转换，因此其具体的实现方法也略有不同。

(2) 十进制小数转换为二进制小数

① 减权定位法

与整数转换所用的减权定位法相似，也是自高位起按位权逐位分离。但转换后得到的二进制小数其位数可能很长，究竟在小数点之后取多少位呢？这要根据所规定的字长或实际需要的精度来决定。

【例 2-9】 将 $(0.635)_{10}$ 转换为二进制小数。

| 减权比较 | X_i | 位权 |
|-------------------------|-------|-------|
| $0.635 - 0.5 = 0.135$ | 1 | 0.5 |
| $0.135 < 0.25$ | 0 | 0.25 |
| $0.135 - 0.125 = 0.010$ | 1 | 0.125 |

转换结果： $(0.635)_{10} = (0.101\cdots)_2$ 。

② 乘基取整法

将式(2-2)的小数部分单独写出：

$$S = X_{-1} 2^{-1} + X_{-2} 2^{-2} + \cdots + X_{-m} 2^{-m}$$

等式两边同乘以基数2，得到下式：

$$2S = X_{-1} + (X_{-2} 2^{-1} + \cdots + X_{-m} 2^{-m+1}) \quad (2-6)$$

显然，式(2-6)的括号内仍为小数；若 $2S$ 出现整数部分，则表明 X_{-1} 为1；若 $2S$ 仍为小数，则表明 X_{-1} 为0。继续对小数部分乘以基数2，根据各次乘积是否出现整数部分，依次判定 $X_{-2} \sim X_{-m}$ 。

由此得到乘基取整法的转换方法如下：将待转换的十进制小数乘以基数 2，所得的整数部分就是二进制小数的高位值；继续对所余小数部分乘以基数 2，所得的整数部分就是次高位值；如此继续，直到乘积的小数部分已为 0，或已满足所需精度要求为止。

【例 2-10】 将 $(0.625)_{10}$ 转换为二进制小数。

| 小数部分乘以 2 | 整数部分 |
|-------------------------|------|
| $0.625 \times 2 = 1.25$ | 1 |
| $0.25 \times 2 = 0.5$ | 0 |
| $0.5 \times 2 = 1$ | 1 |

转换结果： $(0.625)_{10} = (0.101)_2$ 。

前面已经指出，将十进制数转换为二进制数采取的基本思路是：按二进制位权逐位分离出二进制数位，然后进行转换。转换有两种方法。一是在减权定位法中，不论是整数还是小数，都是从二进制数高位起逐位分离。二是基于与基数的乘、除运算，就要从小数点起分别向两边延伸，即分成整数和小数分别转换。对整数部分是从二进制数低位开始，通过除基取余进行分离。对小数部分是从二进制数高位开始，通过乘基取整进行分离。

(3) 二进制整数转换为十进制整数

① 按权相加法

将二进制数展开成多项式求和的形式，即引入各位的权值，按各位的权与系数求出各项的数值，然后相加，就得到转换结果。

【例 2-11】 $(1011)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = (11)_{10}$ 。

这种方法比较直观，适于手算。但各位的权值不同，所以各步操作不统一，程序实现较烦琐。

② 逐次乘基相加法

将式 (2-2) 的整数部分改写：

$$\begin{aligned} S &= X_n 2^n + X_{n-1} 2^{n-1} + \cdots + X_1 2^1 + X_0 2^0 \\ &= \{[(X_n \times 2 + X_{n-1}) \times 2 + X_{n-2}] \times 2 + \cdots\} \times 2 + X_0 \end{aligned} \quad (2-7)$$

从式 (2-7) 可以推导出逐次乘基相加法的转换过程如下：从二进制的最高位开始，乘以基数 2，然后与次高位也就是相邻低位相加；所得结果再乘以基数 2，再与相邻低位相加；如此继续，直到加上最低位为止，所得就是最后结果。

【例 2-12】 将 $(1011)_2$ 转换为十进制数。

$$\begin{aligned} 1 \times 2 + 0 &= 2 \\ 2 \times 2 + 1 &= 5 \\ 5 \times 2 + 1 &= 11 \end{aligned}$$

转换结果： $(1011)_2 = (11)_{10}$ 。

从式 (2-2) 可以看出，相邻两位权值之比等于基数。逐次乘基法就是根据这一点，对高位乘以基数之后就能跟相邻低位相加了，而逐次乘以基数后就能让各位恢复原来的权值。与直接按位权实现各位相加的方法相比，逐次乘基相加法的各步操作统一，更适于编程实现。

(4) 二进制小数转换为十进制小数

① 按权相加法

小数转换的按权相加法与整数转换的按权相加法相同。

【例 2-13】 $(0.1011)_2 = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} = (0.6875)_{10}$ 。

② 逐次除基相加法

将式 (2-2) 的小数部分改写为

$$S = X_{-1}2^{-1} + X_{-2}2^{-2} + \cdots + X_{-m}2^{-m} \\ = 2^{-1}\{X_{-1} + 2^{-1}[X_{-2} + \cdots + 2^{-1}(X_{-m+1} + 2^{-1}X_{-m})]\} \quad (2-8)$$

从式(2-8)可以推导出逐次除基相加法的转换过程如下：从二进制数的最低位开始，除以基数 2 之后与次低位（也就是相邻高位）相加，所得结果再除以 2；其结果继续与相邻高位相加，再除以 2；如此继续，直到与小数点后的第一位相加并除以 2 为止。

【例 2-14】 将 $(0.1011)_2$ 转换为十进制小数。

$$(1 + 1 \times 2^{-1}) \times 2^{-1} = 0.75$$

$$(0 + 0.75) \times 2^{-1} = 0.375$$

$$(1 + 0.375) \times 2^{-1} = 0.6875$$

转换结果： $(0.1011)_2 = (0.6875)_{10}$ 。

同样根据“相邻两位权值之比等于基数值”，与整数不同，小数部分位权值 2^i 中的 i 是负值，因此小数部分的转换采取从最低位开始逐次除基相加方法。除以基数 2 后就可以与相邻高位相加，而逐次除以基数后就能让各位恢复原来的权值。

至此，我们对每种转换都提供了两种方法：第一种方法比较直观，适于手算；第二种方法各步操作统一，更适于编程实现。当然，在具体实现上允许有一些变化。

2.1.2 带符号数的表示

首先，我们来定义两个术语：真值、机器数。

在日常的书写习惯中，往往用正、负符号加绝对值表示数值，用这种形式表示的数值称为真值。例如，用十进制数表示的真值 139、-139，用二进制数表示的真值 1011、-1011 等。编程时常采用真值形式表示数值。

在计算机内部使用的，连同数符一起数字化了的数称为机器数。在由真值转换为计算机硬件能够直接识别、处理的机器数时，需要解决三个问题：① 只能采用二进制数，每位数码非 0 即 1；② 将符号位数字化，如用 0 表示正号，用 1 表示负号；③ 采用什么编码方法表示数值。相应地，机器数可以有四种码制：原码、补码、反码和移码，其中比较最常用的是补码和原码，CPU 硬件一般都支持补码运算和原码运算。

1. 原码表示法

原码表示法约定：让数码序列的最高位为符号位，符号位为 0 表示该数为正，为 1 表示该数为负；数码序列的其余部分为有效数值，用二进制数绝对值表示。换句话说，原码表示法是数码化的符号位加上数的绝对值。根据这一约定，可分别得到纯小数（定点小数）与纯整数（定点整数）的原码定义式如下。

(1) 若定点小数的原码序列为 $X_0.X_1X_2\cdots X_n$ ，则

$$[X]_{\text{原}} = \begin{cases} X & 0 \leq X < 1 \\ 1 - X = 1 + |X| & -1 < X \leq 0 \end{cases} \quad (2-9)$$

式中， X 表示真值， $[X]_{\text{原}}$ 为用原码表示的机器数，可简写为 $X_{\text{原}}$ 。当 X 为正时， $X_{\text{原}}$ 与 X 相同，即 $X_{\text{原}}$ 中的符号位为 0，再加上小数部分的绝对值。当 X 为负时， $X_{\text{原}} = 1 + |X|$ ，即 $X_{\text{原}}$ 中的符号位为 1，再加上小数部分的绝对值。在小数的原码中，符号位 X_0 固定为小数点左边的一位。式(2-9)中为符号位被赋予位权 2^0 。

【例 2-15】 若 $X = +0.1011$ ，则 $X_{\text{原}} = 0.1011$ 。

【例 2-16】若 $X = -0.1011$ ，则 $X_{\text{原}} = 1.1011$ 。

(2) 若定点整数的原码序列为 $X_n X_{n-1} \cdots X_0$ ，其中 X_n 表示符号位，则

$$[X]_{\text{原}} = \begin{cases} X & 0 \leq X < 2^n \\ 2^n - X = 2^n + |X| & -2^n < X \leq 0 \end{cases} \quad (2-10)$$

当 X 为正时， $X_{\text{原}}$ 与 X 相同，即符号位为 0。当 X 为负时， $X_{\text{原}} = 2^n + |X|$ ，即符号位 X_n 为 1 再加上整数部分的绝对值。注意，在整数的原码中，符号位 X_n 是数码序列中最高位，对应于 $n+1$ 位整数数码序列，式 (2-10) 中为符号位 X_n 赋予位权 2^n 。

【例 2-17】若 $X = +1011$ ，则 5 位字长的 $X_{\text{原}} = 01011$ ，而 8 位字长的 $X_{\text{原}} = 00001011$ 。

【例 2-18】若 $X = -1011$ ，则 5 位字长的 $X_{\text{原}} = 11011$ ，而 8 位字长的 $X_{\text{原}} = 10001011$ 。

对于某一种计算机，硬件的基本字长固定，即对机器数位有约定。在小数中，符号位是最高位，位于小数点的左边；有效数值占据高位部分，紧跟小数点之后；如果机器数的字长大于有效数值位数，则所余低位中数码为 0，书写时可以略去，参见例 2-15 和例 2-16。对于整数，符号位仍是最高位，但有效数值占据低位部分，如果机器数的字长大于有效数值位数，则所余的高位段（符号位与有效数值之间）必须补充数码 0，参见例 2-17 和例 2-18。

(3) 讨论

在介绍了补码之后，我们将在图 2-2 中沿数轴分布列出真值、原码、补码三者之间的对应关系和一些典型值。

分析式 (2-9)、式 (2-10) 和图 2-2，可以得出以下一些结论。

① 在原码表示中，真值 0 可以有两种不同的表示形式，可分别称为 +0 与 -0。以小数为例：

$$[+0]_{\text{原}} = 0.00 \cdots 0$$

$$[-0]_{\text{原}} = 1.00 \cdots 0$$

它们的真值含义相同。

② 对于小数原码，表示范围： $-1 < X < 1$ ；对于整数原码 $X_n X_{n-1} \cdots X_0$ ，表示范围： $-2^n < X < 2^n$ 。

③ 符号位不是数值的一部分，是人为地约定“0 正 1 负”。所以在原码运算中需将符号位与有效数值部分分开处理，也就是取数的绝对值进行运算（又称为无符号数运算），而把符号位单独处理，这一点请大家务必注意。

因为数值是用绝对值表示的，所以原码表示很直观，用原码实现乘除运算也比较方便。但是用原码做加、减运算比较复杂，它的实际操作要由操作性质（加还是减）和两个数的数符综合决定，如 $3 - (-2) = ?$ 表面上是要做减法运算，但由于减数是负数，所以实际操作是 $3 + 2$ 。

2. 补码表示法

为了克服原码表示法在加、减运算中的缺点，引入了补码表示法，并以此作为加、减运算的基础。引入补码表示法的目的是：让符号位也作为数值的一部分直接参与运算，以简化加、减运算的规则，同时又能化减为加。下面通过两个例子说明补码的思想。

第一个例子是时钟。时钟以 12 为一个计数循环，在有模运算中称为“以 12 为模”。13 点舍去模 12 后，就是 1 点。从 0 点位置出发，沿反时针方向将时针拨动 -1 格（即 -1 点），等同于沿顺时针方向拨动 11 格（即 11 点）。换句话说，在以 12 为模的前提下，-1 可以映射为 +11，如图 2-1 所示。由此我们得到启发：在有模运算中，一个负数可以用一个与它互为补码的正数来代替。

第二个例子是对角度的表示方法。在讨论三角函数时，我们将 360° 分为 $0 \sim +180^\circ$ （正域）和 $0 \sim -180^\circ$ （负域），以 360° 为一个计数循环。在以 360° 为模的前提下， 370° 可认为是 10° ，即

$370^\circ - 360^\circ = 10^\circ$ ；而 -30° 可以映射为 330° ，即 $360^\circ - 30^\circ = 330^\circ$ 。这个例子中，模值与正、负域范围之间的关系与计算机中补码表示非常相似。补码定点小数的表示范围约为 $-1 \sim +1$ ，以 2 为模。

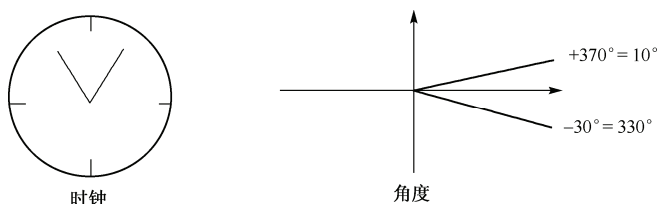


图 2-1 补码思想示意图

计算机的寄存器与运算部件都有一定的字长限制，如一定位数的计数器，在计满后会产生溢出，又从头开始计数。产生溢出的量就是计数器的模，相当于时钟例子中的模 12。因此，计算机中的运算也是一种有模运算，如在以 2 为模的前提下表示 $-1 \sim +1$ 之内的小数，也可以实现正、负数之间的互补性映像。

(1) 补码定义

① 补码的统一定义式：

$$[X]_{\text{补}} = M + X \pmod{M} \quad (2-11)$$

式中，模为 M ， X 是真值； $[X]_{\text{补}}$ 是数 X 的补码，或称为 X 对模 M 的补数，可简称为 $X_{\text{补}}$ 。

若 $X \geq 0$ ，式 (2-11) 中的模 M 可作为溢出量舍去，如同在时钟一例中舍去模 12 一样，则 $X_{\text{补}} = X$ 。若 $X < 0$ ，则 $X_{\text{补}} = M + X = M - |X|$ 。

式 (2-11) 是一个对正负数都适用的统一定义式，由它可以推导出定点小数与定点整数的补码定义如下。

② 定点小数的补码定义式：若定点小数的补码序列为 $X_0.X_1X_2\cdots X_n$ ，其溢出量为 2^1 （注意：符号位 X_0 的权值是 2^0 ），因此以 2 为模。

$$[X]_{\text{补}} = \begin{cases} X & 0 \leq X < 1 \\ 2 + X = 2 - |X| & -1 \leq X < 0 \end{cases} \quad (2-12)$$

【例 2-19】 若 $X = 0.1011$ ，则 8 位字长的 $X_{\text{补}} = 0.1011000$ 。

因为 X 是正数，所以 $X_{\text{补}} = X$ 。对于 8 位字长，低位补 0；对于正数，补码与原码相同。

【例 2-20】 若 $X = -0.1011$ ，则 $X_{\text{补}} = 2 - 0.1011 = 1.0101$ ，写成 8 位字长， $X_{\text{补}} = 1.0101000$ 。

因为 X 是负数，所以 $X_{\text{补}} = 2 - |X|$ ，故得到的结果中符号位为 1，但它是根据式 (2-12) 计算得到的。相应地，小数点之后的部分在形式上与原码、绝对值不同。

③ 定点整数的补码定义式：若定点整数的补码序列为 $X_nX_{n-1}\cdots X_1X_0$ ，即连同符号位有 $n + 1$ 位，其溢出量为 2^{n+1} （注意：符号位 X_n 的权值是 2^n ），因此以 2^{n+1} 为模。

$$[X]_{\text{补}} = \begin{cases} X & 0 \leq X < 2^n \\ 2^{n+1} + X = 2^{n+1} - |X| & -2^n \leq X < 0 \end{cases} \quad (2-13)$$

【例 2-21】 若 $X = 1011$ ，则 5 位字长的 $X_{\text{补}} = 01011$ ，而 8 位字长的 $X_{\text{补}} = 00001011$ 。当机器数字长超出有效数值的位数时，高位部分补 0。

【例 2-22】 若 $X = -1011$ ，则 5 位字长的 $X_{\text{补}} = 100000 - 1011 = 10101$ ，而 8 位字长的 $X_{\text{补}} = 100000000 - 1011 = 11110101$ 。注意，当机器数字长超出有效数值的位数时，高位部分补 1。

(2) 由真值、原码转换为补码

通常，在编程时用真值来表示数值，经过编译、解释后转换成用原码或补码表示的机器数，这就存在一个转换方法的问题。如果由真值转换为补码，可以先按真值写出原码（即加上符号位），

再由原码转换为补码。虽然我们可以根据补码定义式由真值求得补码，但只要比较一下原码与补码之间在形式上的差异，就可以找到几种更简便实用的方法。

① 正数的补码表示与原码相同。

【例 2-23】 若 $X_{\text{原}} = 0.1010$ ，则 $X_{\text{补}} = 0.1010$ 。

② 负数原码转换为负数补码的方法之一：符号位保持为 1 不变，其余各位先变反，然后在末位加 1（在定点小数中，末位加 1 相当于数值加 2^{-n} ）。这一方法可以简称为“变反加 1”。

【例 2-24】 若 $X_{\text{原}} = 1.1010$ ，求 $X_{\text{补}} = ?$

$$\begin{array}{rcl} X_{\text{原}} = & & 1.1010 \\ \text{尾数变反} & & 1.0101 \\ \text{末位加 1} & + & \underline{\quad 1 \quad} \\ X_{\text{补}} = & & 1.0110 \end{array}$$

注：通常将除符号位之外的有效数值部分称为尾数。

③ 负数原码转换为负数补码的方法之二：符号位保持为 1 不变，尾数部分自低位向高位，第一个 1 及其以前的各低位 0 都保持不变，以后的各高位则按位变反。

【例 2-25】 若 $X_{\text{原}} = 1.1010$ ，求 $X_{\text{补}} = ?$

$$\begin{array}{rcl} X_{\text{原}} = 1. & 10 & 10 \\ X_{\text{补}} = \underline{1.} & \underline{01} & \underline{10} \\ & \text{不变} & \text{变反} \quad \text{不变} \end{array}$$

这两种方法所获得的结果与根据补码定义式（2-13）所得的结果是一致的，但更简便，可作为逻辑实现的依据。计算机中更多地采用第②种方法实现求补，即用寄存器的反相输出，通过加法器在末位加 1，从而可并行地实现求补。也有些专用的串行求补逻辑线路采用第③种方法。

（3）由补码表示转换为原码与真值

计算机运算结果常为补码表示，但负数的补码不直观，需要转换为真值形式或比较直观的原码表示。如前所述，正数补码与原码相同，不需要转换。由负数补码转换为原码则可采取上述两种方法之一，对补码再次进行补码表示即可。

【例 2-26】 若 $X_{\text{补}} = 1.0110$ ，求 $X_{\text{原}}$ 与真值。

$$\begin{array}{rcl} X_{\text{补}} = & & 1.0110 \\ \text{尾数变反} & & 1.1001 \\ \text{末位加 1} & + & \underline{\quad 1 \quad} \\ X_{\text{原}} = & & 1.1010 \\ \text{真值} & & -0.1010 \end{array}$$

【例 2-27】 同上题，用另一种方法求 $X_{\text{原}}$ 与真值。

$$\begin{array}{rcl} X_{\text{补}} = 1. & 01 & 10 \\ X_{\text{原}} = \underline{1.} & \underline{10} & \underline{10} \\ & \text{不变} & \text{变反} \quad \text{不变} \\ \text{真值} & & -0.1010 \end{array}$$

以上 4 个例子都是以小数为例讨论转换方法，整数的转换方法也与之相同，请读者自己通过实例分析。

（4）讨论

我们沿数轴列出真值、原码、补码三者的一些典型值，如图 2-2 所示。该图中给出了一些典型的真值，并在对应位置标明原码与补码的数码序列。

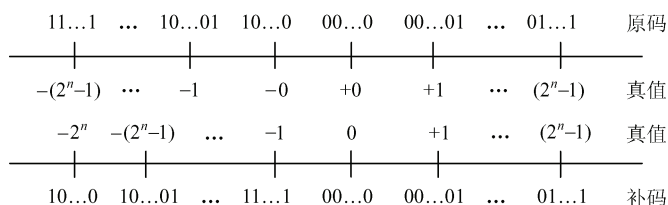


图 2-2 真值、原码、补码之间的对应关系

根据式 (2-12)、式 (2-13) 和图 2-2，我们可以得出以下一些结论。

① 在补码表示中仍以最高位作为符号位，“0 正 1 负”，这点与原码相同。但补码的符号值由补码定义式计算而得，它是数值的一部分，可以与尾数一起直接参与运算，不需要单独处理。例如，负小数补码中符号位 X_0 为 1，这个 1 是真值 X （负）加模 2 后产生的。

② 在补码表示中，数值 0 只有一种表示，即 $00\cdots00$ 。图 2-2 清楚地表明了这点。请注意补码定义式 (2-12)、式 (2-13) 与原码定义式 (2-9)、式 (2-10) 在数域划分上的一点细微差别。例如，式 (2-9) 中对原码小数负数域定义是 $-1 < X \leq 0$ ，而式 (2-12) 中对补码小数负数域则定义为 $-1 \leq X < 0$ 。也就是说：在原码中有“-0”，而补码中没有“-0”。

③ 从补码定义式与原码定义式数域划分的比较中还可发现：负数补码的表示范围比原码稍宽一点，即多一种组合。如图 2-2 所示，整数原码表示中的绝对值最大负数是 $-(2^n-1)$ ，而补码表示中的绝对值最大负数是 -2^n ，其代码序列是 $10\cdots00$ 。

④ 将负数 X 的真值与其补码 $X_{\text{补}}$ 作一映射图，可以进一步看出：负数补码表示的实质是将负数映射到正数域，因而可实现化减为加，达到简化运算的目的。

以定点整数为例，根据补码定义式可以得到如图 2-3 所示的映射关系。由图 2-3 可知：在引入模 2^{n+1} 之后，用 $X_{\text{补}} = 2^{n+1} + X$ 表示负数 X ，相当于将负数 X 向数轴的正向平移 2^{n+1} ，使负数 $X_{\text{补}}$ 映射到正域之中。于是，减一个正数（即加一个负数）被转化为加一个正数，这个正数就是负数补码的映射值。 $X_{\text{补}}$ 中的符号位是映射值中的最高数位，因此在补码运算中符号位就像数值的一部分一样可以直接参与运算。

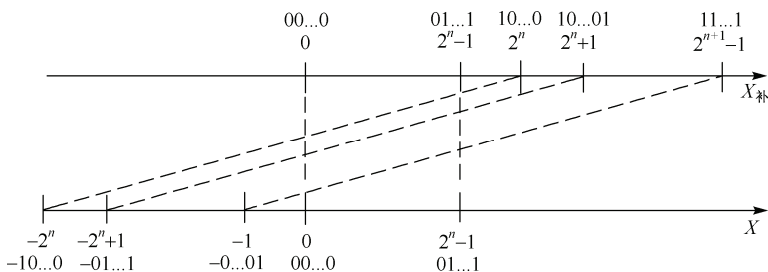


图 2-3 补码与真值间的映射图

3. 反码表示法

除前面介绍的原码与补码外，还有一种机器数的表示方法即反码，约定如下：正数的反码表示与原码相同；负数反码的符号位为 1，尾数由原码尾数逐位变反。对比由原码转换为补码的方法，不难看出，在形式上反码跟补码的区别就是末位少加一个 1，由补码定义推得反码定义如下：

若定点小数的反码序列为 $X_0.X_1X_2\cdots X_n$ ，则

$$[X]_{\text{反}} = \begin{cases} X & 0 \leq X < 1 \\ (2 - 2^{-n}) + X & -1 < X \leq 0 \end{cases} \quad (2-14)$$

若定点整数的反码序列为 $X_n X_{n-1} \cdots X_0$ ，则

$$[X]_{\text{反}} = \begin{cases} X & 0 \leq X < 2^n \\ (2^{n+1} - 1) + X & -2^n < X \leq 0 \end{cases} \quad (2-15)$$

【例 2-28】若 $X_{\text{原}} = 0.1010$ ，则 $X_{\text{反}} = 0.1010$ 。

【例 2-29】若 $X_{\text{原}} = 1.1010$ ，则 $X_{\text{反}} = 1.0101$ 。

反码也能化减为加，但现在较少直接采用反码运算，所以本节不详细讨论。

2.1.3 定点数与浮点数

实际的数可能既有整数部分又有小数部分。众所周知，在进行加减运算时需要先将两个数的小数点位置对准，然后才能加、减。这就提出了一个如何表示小数点位置的问题。根据小数点位置是否固定，在计算机中，数的格式又可分为定点表示、浮点表示两类。

1. 定点表示法

在计算机中，小数点位置固定不变的数叫做定点数。为了处理方便，一般只采取三种简单的约定，相应地有三种类型的定点数。

(1) 无符号定点整数

无符号定点整数由于没有符号位，全部数位都被用来表示数值，因此它的表示范围比带符号定点整数更大，且它的小数点隐含在最低位之后，在数码序列中并不存在。

对于某一种数的表示方法，我们关心它的两项指标：一是表示范围，即这种方法能表示多大的数（正、负两个方向）；二是分辨率，即精细的程度（精度）。这就好像一把尺子一次测量的范围由其长度决定，而尺子的精度由它的最小刻度（是厘米还是毫米）来决定一样。在分析出有关的典型值之后就可以得到这两项指标。

若无符号定点整数的代码序列为 $X_n X_{n-1} \cdots X_1 X_0$ ，即 $n+1$ 位正整数，则有：

| 典型值 | 真值 | 代码序列 |
|--------|-------------|---------|
| 最大正数 | $2^{n+1}-1$ | 11...11 |
| 最小非零正数 | 1 | 00...01 |

由于是正数，原码形式与补码形式相同，两者之间没有区别，也不需要符号位，其表示范围为 $0 \sim (2^{n+1}-1)$ ，相应的分辨率为 1。

对于正数域，由 0 至最大正数这一范围就是表示范围。非零最小正数这一典型值就是其分辨率，它表明了这种表示方法的绝对精度。

(2) 带符号定点整数

带符号定点整数是纯整数，小数点在最低位之后，最高位为符号位。常用补码表示，也有采用原码表示的，因此在列出典型值与表示范围时需分成原码与补码来讨论。

若带符号定点整数的代码序列是 $X_n X_{n-1} \cdots X_1 X_0$ ，其中 X_n 是符号位，则有：

| 典型值 | 真值 | 代码序列 |
|-----------|------------|---------|
| 原码绝对值最大负数 | $-(2^n-1)$ | 11...11 |
| 原码绝对值最小负数 | -1 | 10...01 |
| 原码最小非零正数 | +1 | 00...01 |
| 原码最大正数 | 2^n-1 | 01...11 |
| 补码绝对值最大负数 | -2^n | 10...00 |
| 补码绝对值最小负数 | -1 | 11...11 |

| | | |
|----------|---------|---------|
| 补码最小非零正数 | +1 | 00...01 |
| 补码最大正数 | 2^n-1 | 01...11 |

前面对原码与补码的分析讨论，很大程度上在这些典型值中得到体现。由绝对值最大的负数到最大正数，就是该码制定点数的表示范围。因此得到：

原码定点整数表示范围： $-(2^n-1) \sim (2^n-1)$

补码定点整数表示范围： $-2^n \sim (2^n-1)$

原码、补码定点整数分辨率：1

(3) 带符号定点小数

带符号定点小数是纯小数，用原码或补码表示，若代码序列为 $X_0X_1X_2\cdots X_n$ ，最高位 X_0 是符号位，小数点位置在符号位之后。在原码中， $X_1\cdots X_n$ 是数值的有效部分，常称为尾数，并将 X_1 称为最高数位或最高有效位，在补码中也沿用这种叫法。

| 典型值 | 真值 | 代码序列 |
|-----------|---------------|----------|
| 原码绝对值最大负数 | $-(1-2^{-n})$ | 1.1...11 |
| 原码绝对值最小负数 | -2^{-n} | 1.0...01 |
| 原码最小非零正数 | 2^{-n} | 0.0...01 |
| 原码最大正数 | $1-2^{-n}$ | 0.1...11 |
| 补码绝对值最大负数 | -1 | 1.0...00 |
| 补码绝对值最小负数 | -2^{-n} | 1.1...11 |
| 补码最小非零正数 | 2^{-n} | 0.0...01 |
| 补码最大正数 | $1-2^{-n}$ | 0.1...11 |

相应地，对于 $n+1$ 位定点小数 $X_0.X_1X_2\cdots X_n$ ，其表示范围与分辨率为：

原码定点小数表示范围： $-(1-2^{-n}) \sim (1-2^{-n})$ ；

补码定点小数表示范围： $-1 \sim (1-2^{-n})$ ；

分辨率： 2^{-n} 。

定点数的小数点位置是固定的，不需要设置专门的硬件或数位来表示它。换句话说，小数点本身并不存在，在书写定点小数的格式中标注小数点仅仅是为了醒目而已，提醒你这是小数。所以，上述三种定点数是在程序中的一种隐含约定，在硬件上并无区别。编程者根据自己的需要，在程序中自行约定选择哪一种定点数格式。

不难想到，如果某个数据既有整数又有小数，要将它规范为某种定点数，就需要在程序中设置比例因子，才能将它缩小为定点小数或扩大为定点整数。运算后，再根据比例因子与实际经历的运算操作，将所得到的运算结果还原为实际值。

大家可能注意到，在描述小数序列时我们采用的顺序是 $X_0X_1X_2\cdots X_n$ ，即 X_0 为最高位而 X_n 为最低位。在描述整数序列时我们采用的顺序是 $X_nX_{n-1}\cdots X_0$ ，即 X_0 为最低位， X_n 为最高位。这是因为在这两种定点数中 X_0 位的权值都是 2^0 ，所以这种顺序可适于通式描述，与位数无关。在常用计算机中，大多采取以定点整数为参考体系的定义方法，如定义 8 位数据线为 $D_7\sim D_0$ （低），16 位地址线为 $A_{15}\sim A_0$ （低）。但是在讨论运算方法时又常以小数为对象，再将所得到的运算方法推广到整数。这是因为定点小数补码以 2^1 为模，与位数无关，而定点整数的模 2^{n+1} 则与位数有关。

定点数的表示范围有限，如果运算结果超出表示范围，称为溢出。大于最大正数，称为正溢。沿负的方向超出绝对值最大负数（或描述为小于定点数的最小值），称为负溢。如果比例因子选择不当，如在变为定点小数时缩小比例不足，运算就可能产生溢出。因此，计算机硬件应具有溢出判断功能，一旦产生溢出就可以立即转入溢出处理，调整比例因子。反之，在设置比例因子时如果缩小比例过大，将会降低精度。

定点数比较简单，实现定点运算的硬件成本比较低。但在有限位数的定点数中，表示范围与精度这两项指标不易兼顾，选取比例因子的办法也比较麻烦。

2. 浮点表示法

通过前面的分析可以看出，不论是哪一种定点数，如果位数固定，则它的表示范围和分辨率两项指标也就固定不变，不能根据实际需要而灵活改变，只能依靠外设比例因子的办法。如果采取数字的科学表示法，并且将比例因子作为数的一部分也包含在数的内部，就既能够按照实际需要表示数的大小，又能具有足够的相对精度，由此引出了数的浮点表示法。浮点数是一种小数点位置不固定可随需要浮动的数。

(1) 浮点数的表示形式（原理性）

先将数写成一种比例因子与尾数相乘的形式，而且比例因子采用指数形态。

$$N = \pm R^E \times M \quad (2-16)$$

式中， N 为真值， R^E 为比例因子， M 是尾数。对于某种浮点格式， R 固定不变且隐含约定，因此浮点数代码序列中只需分别给出 E 和 M 这两部分（连同它们的符号）。相应地，浮点数的原理性表达格式就可以被表示成如图2-4所示的样式。

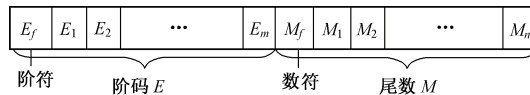


图 2-4 浮点数表示格式示例

E 是阶码，也就是比例因子 R^E 的指数值，为带符号定点整数，可用补码或移码表示（有关移码的概念将在后面介绍）。若阶码 E 为正，表明尾数 M 将被扩大若干倍；若 E 为负，表明 M 将被缩小若干倍。

R 是阶码的底，与尾数 M 的基数相同。例如，尾数为二进制，基数为 2，则选择 $R=2$ 。如前所述，对于某一种浮点数据格式， R 是隐含约定的一个值，因此不占用代码位。

式（2-6）中的 M 是浮点数的尾数，它是一个带符号的定点原码或补码小数。为了充分利用尾数部分的有效位数，使表示精度尽可能高，以及确保任何一个数用浮点数形式表示时其尾数的代码表示具有唯一性，故一般都对尾数 M 有规格化要求。

如果以 $R=2$ 为基底，若浮点数用原码来表示，则尾数规格化的要求是使尾数的绝对值满足条件： $1/2 \leq |M| < 1$ ，而此时满足该条件的规格化尾数其最高有效位将始终为 1。

同样还是以 $R=2$ 为基底，若浮点数用补码来表示，则尾数规格化的要求是使尾数的绝对值满足条件： $-1 \leq M < -1/2$ 或 $1/2 \leq M < 1$ ，此时规格化尾数的符号位与最高有效位刚好相反。具体而言，当 $-1 \leq M < -1/2$ 时，规格化尾数的最高有效位将为 0；当 $1/2 \leq M < 1$ 时，规格化尾数的最高有效位将为 1。

对于正数，无论是用原码还是补码来表示，其规格化的特征是 $M_1=1$ ；对于原码表示的负数，其规格化的特征也是 $M_1=1$ 。但是，对于补码表示的负数，其规格化的特征却是 $M_1=0$ 。由此可见，我们可以根据尾数的正负性质、原码还是补码表示以及最高有效位 M_1 是 1 还是 0，来快速判定该尾数是否属于已经被规格化的尾数。

在分析尾数规格化时， -1 和 $-1/2$ 这两个特殊取值需要特别注意。用原码来表示尾数时， M 不可能等于 -1 ，而用补码来表示时 M 却可以等于 -1 ，根据规格化的定义可知：原码表示时 -1 不是一个规格化的尾数，而补码表示时 -1 却是一个规格化的尾数。同理，对原码表示而言 $-1/2$ 是一个规格化的尾数，但对补码而言 $-1/2$ 却不是一个规格化的尾数。

此外还应当注意，数符 M_f 决定了浮点数的正负性质，而阶符 E_f 仅决定阶码自身的正负性质，同时也表明阶符为正时需将尾数扩大，为负时需将尾数缩小。

(2) 移码（增码）

刚才提到，有的浮点数阶码是用移码来表示的。移码是一种专门用于浮点数阶码表示的码制，采用这种表示方法可以更方便地比较两浮点数阶码的大小。

若用移码表示的浮点数阶码共有 $m+1$ 位，其代码序列为 $X_m X_{m-1} \cdots X_1 X_0$ ，则

$$X_{\text{移}} = 2^m + X \quad -2^m \leq X < 2^m \quad (2-17)$$

上式中， X 是阶码的真值， 2^m 是数字位 X_m 的位权。因此 $X_{\text{移}}$ 相当于将真值 X 沿数轴正向平移 2^m ，所以形象地将其称为移码，也可以理解成将 X 增加 2^m ，故又叫做增码。

【例 2-30】 某补码表示的浮点数阶码（连同 1 位阶符）共 8 位，则阶码的真值表示范围为： $-128 \leq X \leq 127$ 。若将阶码的真值 X 用移码来表示，可得到 $X_{\text{移}} = 2^7 + X$ ，即 $0 \leq X_{\text{移}} \leq 255$ 。

此浮点数阶码的真值 X 与长度为 8 位的原码、移码和补码之间的对应关系如表 2-2 所示。

表 2-2 真值、移码、补码对照表

| 真值 X (十进制) | $X_{\text{原}}$ | $X_{\text{移}}$ | $X_{\text{补}}$ |
|--------------|-------------------|----------------|----------------|
| -128 | 超出表示范围 | 00000000 | 10000000 |
| -127 | 11111111 | 00000001 | 10000001 |
| ... | ... | ... | ... |
| -1 | 10000001 | 01111111 | 11111111 |
| 0 | 00000000/10000000 | 10000000 | 00000000 |
| +1 | 00000001 | 10000001 | 00000001 |
| ... | ... | ... | ... |
| +127 | 01111111 | 11111111 | 01111111 |

从表 2-2 中我们可以直观地看出：

- ① 移码符号位为 0 时， X 为负，为 1 时， X 为正。这一点与原码、补码、反码相反。
- ② 除符号位相反之外，移码的其余各位与补码相同。这是由于 X 平移 2^7 得到了移码，而平移了 2^8 （模值）才能得到补码。
- ③ 让 X 从 -128 逐渐增至 +127，相应地， $X_{\text{移}}$ 从 00...00 逐渐变化至 11...11，呈递增状。由此可见，采用移码能更直观地比较正、负阶码的大小，如 +1 与 -127 之间的比较。

(3) 表示范围与精度

浮点数的原理性格式见图 2-4，其中阶码部分 $m+1$ 位，含 1 位阶符，补码表示，以 2 为底；尾数部分 $n+1$ 位，含 1 位数符，补码表示，规格化。

此时，其表示的典型值如表 2-3 所示。

表 2-3 浮点数的几种典型值

| 典型值 | 浮点数代码 | | 真 值 |
|-------|-----------|------------|--|
| | 阶 码 | 尾 数 | |
| 最小的负数 | 011...111 | 1.00...000 | $(2^{2^n-1}) \times (-1)$ |
| 最大的负数 | 100...000 | 1.01...111 | $(2^{-2^n}) \times (-2^{-1} - 2^{-n})$ |
| 最小的正数 | 100...000 | 0.10...000 | $(2^{-2^n}) \times (2^{-1})$ |
| 最大的正数 | 011...111 | 0.11...111 | $(2^{2^n-1}) \times (1 - 2^{-n})$ |

其表示范围： $-(2^{2^n-1}) \sim (2^{2^n-1}) \times (1 - 2^{-n})$ ；最高分辨率： $(2^{-2^n}) \times (2^{-1})$ 。

对于最大的正数或最小的负数，它的阶码应当是正的最大值，对应的补码代码为 $01\cdots 1$ ，真值是 2^m-1 ，但最小负数的尾数本身也应是最大正数，小数的补码可达 -1 ，其对应的补码为 $1.0\cdots 0$ 。而最大正数的尾数本身也应是最大正数，小数补码可达 $1-2^{-n}$ ，对应的代码为 $0.1\cdots 1$ 。对于最大的负数或最小的正数，它的阶码应当是绝对值最大的负数，其对应的补码为 $10\cdots 0$ ，阶码真值是 -2^m 。注意：规格化尾数 M 的最小非零正值是 2^{-1} （不是 2^{-n} ），而规格化尾数的最大负值是 $-1/2-2^{-n}$ ，并不是 $-1/2$ 。

在表 2-3 中，阶码的二进制代码是用补码来表示的，如果改用移码表示，则代码序列中的阶符与用补码表示时的阶符相反，其余位全部相同。用移码表示不会影响阶码的真值。

现在通过两个例子，对定点表示与浮点表示的两项指标进行对比。

【例 2-31】 若定点整数字长 32 位，含 1 位数符，补码表示，则此定点数的表示范围为： $-2^{31}\sim(2^{31}-1)$ ，分辨率为 1。

【例 2-32】 若浮点数字长 32 位，其中阶码 8 位，含 1 位阶符，补码表示，以 2 为底，规格化的尾数为 24 位，含 1 位数符，补码表示。则该浮点数的表示范围为 $-2^{127}\sim 2^{127}(1-2^{-23})$ ，最高分辨率为 2^{-129} 。

读者可能会问，同样的字长，为什么浮点数的表示范围要比定点数大得多，而且分辨精度也高得多呢？其实，上面给出的分辨率值 2^{-129} 是该浮点格式的最高分辨率，它对应于阶码为绝对值最大负数时。当阶码值增大时分辨率将随之降低（值变大），而阶码值减少时分辨率随之提高（值变小）。例如，当阶码真值为 23 时，分辨率为 1；即尾数改变一个最小量（ 2^{-23} ）时，真值改变值为 1。事实上，当我们要表示一个很大的数值时，分辨精度不会要求很高，如在计算天体之间的距离时就以光年为单位。当需要很高的分辨精度时，数值不会很大，如在半导体芯片加工中，控制精度要达到微米级或纳米级，但芯片尺寸也就是厘米级。由于浮点数的阶码可根据实际需要而变化，在表示较小数值时，可使阶码为绝对值较大的负数，从而获得很高的分辨精度；而在表示精度要求不高的数值时，可使阶码为较大的正数，从而使表示范围很大。

相应地，我们将浮点数的精度分为相对精度与绝对精度。相对精度是尾数本身的分辨率，取决于尾数的位数，在例 2-32 中相对精度为 2^{-n} 。绝对精度 $=R^E \times$ 相对精度，它是浮点数真值的实际分辨率，与阶码值有关。而真值的表示范围则取决于阶码的位数。

阶码的变化使比例因子 R^E 变化，相当于使小数点位置浮动，所以称为浮点数。浮点数不需外设比例因子，容易达到所需的计算精度，精度要求较高时往往采取浮点数运算。一个浮点数由一个定点整数（阶码）和一个定点小数（尾数）组成，因此浮点运算由两组相关的定点运算来实现。

（4）真值与浮点数之间的转换

根据浮点数原理性格式并参照表 2-3 的典型值实例，不难根据一个浮点数代码求出它的真值，或将真值写成浮点数代码。

【例 2-33】 某浮点数格式如图 2-4 所示，字长 32 位；阶码 8 位，含 1 位阶符，补码表示，以 2 为底；尾数 24 位，含 1 位数符，补码表示，规格化。若浮点数代码为 $(A3680000)_{16}$ ，求其真值 N 。

$$(A3680000)_{16} = (10100011, 011010000000\cdots 0)_2$$

$$E = -(1011101)_2 = -(93)_{10}$$

$$M = (0.11010\cdots 0)_2 = (0.8125)_{10}$$

$$N = 2^{-93} \times 0.8125$$

【例 2-34】 按如图 2-4 所示的浮点数格式将 $-(1011.11010\cdots 0)_2$ 写成浮点数代码 F 。

$$N = -(1011.11010\cdots 0)_2$$

$$= -(0.101111010\cdots 0)_2 \times 2^4$$

$$E = (4)_{10} = (00000100)_2$$

$$M_{\text{补}} = (1.010000110\cdots 0)_2$$

$$F = (00000100, 1010000110\cdots 0)_2 = (04A18000)_{16}$$

(5) IEEE754 标准浮点格式

前面讨论的是一种原理性浮点数表示格式，以便于初学者掌握基本原理，在计算机中并不使用那种格式来表示一个浮点数，实用机器的浮点数格式与此有一些差异。

下面简要介绍在当前主流微机中广泛采用的 IEEE754 标准浮点数格式。按照这一标准，有三种浮点格式可供选择，分别称为：短实数、长实数和临时实数，如下：

| | 数符 | 阶码 | 尾数 | 总位数 |
|-------------|----|----|----|-----|
| 短实数（单精度浮点数） | 1 | 8 | 23 | 32 |
| 长实数（双精度浮点数） | 1 | 11 | 52 | 64 |
| 临时实数 | 1 | 15 | 64 | 80 |

以 32 位短浮点数为例，其代码格式如图 2-5 所示。

最高位 S_0 是数符，其后是 8 位的阶码，以 2 为底，采用移码表示，但偏置量为 127。例如，如果某阶码的真值为 1，则该阶码的值应记录为 128（即 $1+127$ ），这点与前述的原理性偏置量 128 不同。其余的 23 位是用原码表示的尾数（符号位 S_0 放在整个浮点数的最高位，不包括在尾数中）为纯小数。此外，IEEE 754 标准还隐含约定尾数的最高数位为 2^0 （即为 1），且这一位并不出现在浮点数的代码序列中，因此尾数实际上相当于是 24 位。相应地，尾数的真值实际上是等于 1 （整数部分） $+M$ （小数部分，23 位）得到的结果。

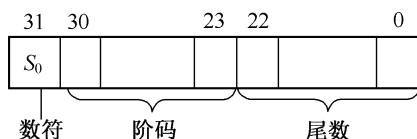


图 2-5 IEEE 754 短浮点数格式

为了确保 IEEE 754 格式浮点数的表示代码具有唯一性，也要求用原码表示的尾数 M 必须满足 $0 \leq |M| < 1$ （即规格化要求）。除此之外，还应注意以下三种特殊情况：

- ① 阶码 E 各位全为 0，且尾数 M 的各位也全为 0 时，则浮点数 $F=0$ 。
- ② 阶码 E 各位全为 1，且尾数 M 的各位却全为 0 时，则浮点数 $F=\pm\infty$ 。
- ③ 阶码 E 各位全为 1，且尾数 M 的各位非全为 0 时，则 F 为无效浮点数。

【例 2-35】 将十进制数 20.59375 转换成符合 IEEE 754 格式的 32 位短浮点数，写出其二进制代码，并转换成 16 进制代码。

首先分别将整数和小数部分转换成二进制数（真值的绝对值）：

$$20.59375 = 10100.10011$$

然后移动小数点，使其在第 1 和第 2 位数字之间：

$$10100.10011 = 1.010010011 \times 2^4$$

小数点左移了 4 位，于是得到： $e=4$ （阶码的真值）。

尾数符号位： $S_0=0$ （正数），阶码表示成移码： $E=4+127=131=10000011$ 。

尾数 $M=010010011$ （这里是原码表示，切勿表示成补码）。

最后得到 IEEE 754 的 32 位浮点数二进制存储格式为：

$$F = (0100\ 0001\ 1010\ 0100\ 1100\ 0000\ 0000\ 0000)_2$$

$$= (41A4C000)_{16}$$

对于 IEEE 754 定义的浮点数，要特别强调几个注意事项：

① IEEE 754 定义的浮点数格式，最高位代码是浮点数的符号位（这与原理格式不同）。

② IEEE 754 定义的浮点数格式，阶码是用移码来表示的，但生成移码时，为了确保移码的最小值是 0，采用的偏移量要比生成标准移码时的偏移量小 1，即生成 8 位阶码时只在真值上加 $2^7-1=127$ （标准移码是加 $2^7=128$ ）。对于长浮点数，以此类推。

③ 无论 32 位还是 64 位的浮点数，尾数均用原码来表示。

④ 在浮点数代码中，尾数 M 是一个绝对值省略了+1 的纯小数，且 $0 \leq |M| < 1$ ，但 M 的实际值应理解成 $1+M$ ，只不过是在表示和存储时，省略了 1 而已。

至此，我们分析了表示数值型数据时所涉及的三方面：进位制、带符号数、小数点，对某一个机器数的具体描述往往需综合这三方面。例如，某数是定点小数，补码表示、二进制数。又如，某数是浮点数，它由两个定点数组成，其中阶码为定点整数、补码表示、二进制数，尾数为定点小数、补码表示、二进制数，等等。

2.2 字符型数据的表示

数据分为数值型和非数值型两大类。工作中可能遇到的非数值型数据非常多，一类是字符及以字符为基础的数据信息，另一类是可以数字化的、范围极其广泛的种种信息，如图像、声音、逻辑信息等。第 1 章已经介绍了信息数字化的基本概念，第 3 章的 3.2 节将介绍指令信息（主要是逻辑信息），第 6 章还将介绍图形信息的表示方法，希望大家能举一反三。本节内容是介绍字符型信息的表示方法。字符是非数值型信息的表示基础，也可以用它来间接表示数值型数据。例如，以字符为基础表示文字，而文字可以描述范围极其广泛的许多信息，其中包含知识在内。又如，用字符与字符串构成程序设计语言，可用来编写程序，程序中包含的数值型数据也是先用字符来描述（如 0~9 和 A~F 等）。

1. ASCII 码

在计算机系统中，无论是信息的输入、传输、处理还是存储或者输出，都涉及字符信息的表示，因此就要为字符制定一种统一的表达格式，才能实现数据的兼容。在国际上广泛采用美国信息交换标准码（American Standard Code For Information Interchange, ASCII）作为字符的表达标准。ASCII 码本来是为信息交换所规定的标准，由于字符数量有限、编码简单，所以输入、存储和计算机内部处理时也往往采用这一标准。

ASCII 字符集中共有 128 种常用字符，其中包含：数字 0~9，大写英文字母，小写英文字母，一些常用的符号如运算符、括号、标点符号、标识符等，还有一些格式控制符。这些字符种类大致满足了各种编程语言、西文文字、常见控制命令等的需要。每个 ASCII 字符自身用 7 位编码，存储器中的 1 字节信息可以存放一个字符的 ASCII 编码。1 字节共 8 位，空出的高位可以用来存放一位奇偶校验位（关于奇偶校验将在第 2.4 节中介绍）。字符与编码的对应关系如表 2-4 所示。

2. 汉字编码简介

与西文不同，汉字字符很多，所以汉字编码比西文编码复杂。在一个汉字信息处理系统的不同部位，需使用几种编码，可分属下述三类：输入码、内部码、交换码。

（1）汉字输入码

对于绝大多数汉字输入人员来说，要直接记住数千个汉字的二进制编码是非常困难的，键盘上也很难将几千个汉字都做成按键。所以需要一些比较直观、方便、快速的汉字输入方法，为此

表 2-4 常见字符的 ASCII 编码

| 十六进制码 | 字符 | 十六进制码 | 字符 | 十六进制码 | 字符 | 十六进制码 | 字符 |
|-------|-----|-------|----|-------|----|-------|-----|
| 00 | NUL | 20 | SP | 40 | @ | 60 | ` |
| 01 | SOH | 21 | ! | 41 | A | 61 | a |
| 02 | STX | 22 | " | 42 | B | 62 | b |
| 03 | ETX | 23 | # | 43 | C | 63 | c |
| 04 | EOT | 24 | \$ | 44 | D | 64 | d |
| 05 | ENQ | 25 | % | 45 | E | 65 | e |
| 06 | ACK | 26 | & | 46 | F | 66 | f |
| 07 | BEL | 27 | ' | 47 | G | 67 | g |
| 08 | BS | 28 | (| 48 | H | 68 | h |
| 09 | HT | 29 |) | 49 | I | 69 | i |
| 0A | LF | 2A | * | 4A | J | 6A | j |
| 0B | VT | 2B | + | 4B | K | 6B | k |
| 0C | FF | 2C | , | 4C | L | 6C | l |
| 0D | CR | 2D | - | 4D | M | 6D | m |
| 0E | SO | 2E | . | 4E | N | 6E | n |
| 0F | SI | 2F | / | 4F | O | 6F | o |
| 10 | DLE | 30 | 0 | 50 | P | 70 | p |
| 11 | DC1 | 31 | 1 | 51 | Q | 71 | q |
| 12 | DC2 | 32 | 2 | 52 | R | 72 | r |
| 13 | DC3 | 33 | 3 | 53 | S | 73 | s |
| 14 | DC4 | 34 | 4 | 54 | T | 74 | t |
| 15 | NAK | 35 | 5 | 55 | U | 75 | u |
| 16 | SYN | 36 | 6 | 56 | V | 76 | v |
| 17 | ETB | 37 | 7 | 57 | W | 77 | w |
| 18 | CAN | 38 | 8 | 58 | X | 78 | x |
| 19 | EM | 39 | 9 | 59 | Y | 79 | y |
| 1A | SUB | 3A | : | 5A | Z | 7A | z |
| 1B | ESC | 3B | ; | 5B | [| 7B | { |
| 1C | FS | 3C | < | 5C | \ | 7C | |
| 1D | GS | 3D | = | 5D |] | 7D | } |
| 1E | RS | 3E | > | 5E | ↑ | 7E | ~ |
| 1F | US | 3F | ? | 5F | - | 7F | DEL |

研究人员已经提出了至少几百种汉字编码方案，较常使用的也有几十种之多。归纳起来，所采用的方法可分为几类：拼音码、字形码、音形结合、具有某些提示和联想功能的方案等。所产生的输入码需要借助输入码与内部码的对照表（称为输入字典），转换成便于加工处理的内部码。

（2）汉字内部码

汉字内部码简称内码，它是计算机内部供存储、处理、传输用的代码。在早期，不同的设计者设计了自己的汉字内码，因而各种计算机使用的汉字内码不统一，这造成了交换汉字信息时的困难。当我国推出了汉字交换码的国家标准之后，于 1990 年又提出了基于 ASCII 代码体系的汉字内码推荐方案，它与国标汉字交换码有一种简单的对应关系，用双字节编码表示一个汉字。

（3）汉字交换码

如前所述，早期的各种汉字系统的内码不统一，因此在各汉字系统之间或汉字系统与通信系统之间进行汉字信息交换（即传输）时，需要制定一种编码标准，即汉字交换码。

首先，我国制定了《信息处理交换用的七位编码字符集》，后来成为国家标准，除个别字符

(如货币符号)外,字符与 ASCII 一致,可视为 ASCII 的中国版本。

1981 年,我国公布了汉字交换码的国家标准《信息交换用汉字编码字符集——基本集》(GB2312—1980),用 2 字节构成一个汉字字符编码,收录了 6763 个汉字字符和 682 个非汉字图形字符(如间隔、标点、运算符、制表符、数字、汉语拼音、拉丁文字母、希腊文字母、俄文字母、日文假名等)。它们排成一个 94×94 的行列矩阵,矩阵的行称为区,列称为位,相应地,每个字符处于某区、某位。而字符的国标交换码与区位号有一个简单的对应关系。

以后,我国又陆续公布了汉字交换码的 5 个辅集,收录了更多的汉字字符。

从几个标准的推出时间看,首先是制定了 ASCII 的中国版,以便与国际接轨,一开始作为部颁标准,后来又成为国家标准。在 ASCII 体系的基础上制定了汉字交换码的国家标准,然后陆续予以补充完善。再往后,在汉字交换码的国标基础上提出了汉字内码的推荐方案。目前,内码推荐方案还不完全等同于汉字交换码国标,但两者之间存在简单的对应关系。

限于篇幅,本节仅仅简要地介绍了三类汉字编码(输入、内部、交换)的基本思想,详细内容请查阅有关资料。

2.3 运算方法

前面部分内容讨论了计算机中数据的表示方式,包括数值型数据和字符型数据。本节将讨论计算机系统中数值型数据的运算方法。数值型数据的各种复杂运算处理都可以根据一定的算法分解为四则运算或基本的逻辑运算,四则运算最终又可以转化成为基本的加法运算,所以本节介绍几种主要的算术运算方法,包括定点四则运算和浮点四则运算方法,并讨论有关算法的逻辑实现。

2.3.1 定点加减运算

在计算机中,带符号的数有原码、补码、反码等几种表示方法。原码加减和补码加减是一切算术运算的基础。但原码加减运算复杂,因为操作数取绝对值运算,实际操作并不仅仅取决于操作码,还与操作数的正负有关,并且可能对运算结果进行修正。补码加减运算则比较简单,因而计算机中基本采用补码加减法。下面主要讨论补码加减运算方法。

1. 补码加减运算的基本关系式

补码加减是指:两个操作数都用补码来表示,连同符号位一起运算,结果也用补码表示。补码加减所依据的基本关系是:

$$(X+Y)_{\text{补}} = X_{\text{补}} + Y_{\text{补}} \quad (2-18)$$

$$(X-Y)_{\text{补}} = X_{\text{补}} + (-Y)_{\text{补}} \quad (2-19)$$

式(2-18)表明:当做加法运算时,可直接将补码表示的两个操作数 $X_{\text{补}}$ 和 $Y_{\text{补}}$ 相加,不必考虑它们的数符是正或负,所得结果即为补码表示的和。

式(2-19)表明:当做减法运算时,可转换为与减数的负数相加,从而化“减”为“加”。其中, $(-Y)_{\text{补}}$ 是 $Y_{\text{补}}$ 的机器负数。由于 $Y_{\text{补}}$ 的值可正可负,故 $(-Y)_{\text{补}}$ 也可正可负。由 $Y_{\text{补}}$ 求 $(-Y)_{\text{补}}$,称为对 $Y_{\text{补}}$ 进行变补,即将 $Y_{\text{补}}$ 连同符号位一起变反后末尾再加 1(无论 $Y_{\text{补}}$ 为正或负)。如果减数 Y 为正,则变补后可转换为加上一个负数;如果减数 Y 为负,则变补后可转换为加上一个正数。

我们举几个例子来验证上述关系式。设机器数有 5 位,包含 1 位符号位。

【例 2-36】 $9+3=12$ $(-9)+(-3)=-12$

| | |
|--|--|
| $\begin{array}{r} 01001 \\ + 00011 \\ \hline 01100 \end{array}$ $9-3=6$ $\begin{array}{r} 01001 \\ + 11101 \text{ (3变补)} \\ \hline 100110 \end{array}$ $3-9=-6$ $\begin{array}{r} 00011 \\ + 10111 \text{ (9变补)} \\ \hline 11010 \text{ (-6补码)} \end{array}$ | $\begin{array}{r} 10011 \text{ (-9补码)} \\ + 11101 \text{ (-3补码)} \\ \hline 110100 \text{ (-12补码)} \end{array}$ <p>↘符号位产生的进位自然丢掉</p> $9-(-3)=12$ $\begin{array}{r} 01001 \\ + 00011 \text{ (-3变补)} \\ \hline 01100 \end{array}$ |
|--|--|

2. 补码加减运算规则

图 2-6 用流程图的形式描述了补码加减算法，其运算规则如下：

- ⊙ 参与运算的操作数用补码表示，符号位作为数的一部分直接参与运算，所得运算结果即为补码表示形式。
- ⊙ 若操作码为加，则两数的补码直接相加。
- ⊙ 若操作码为减，则将减数变补后再与被减数的补码相加。

3. 补码加减运算的逻辑实现

如何用加法器实现补码加减法？图 2-7 给出了补码加、减运算器示意框图。图中有一个 $n+1$ 位的加法器 ($\Sigma_0\sim\Sigma_n$)、提供操作数与暂存结果的寄存器 A 和 B ($n+1$ 位)，运算器功能为 $A\pm B\rightarrow A$ 。现以寄存器级粗框表示，只画出其中一位加减控制逻辑。

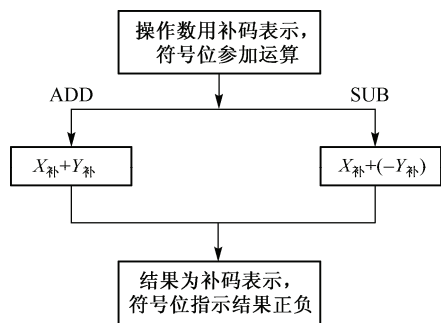


图 2-6 补码加减算法流程

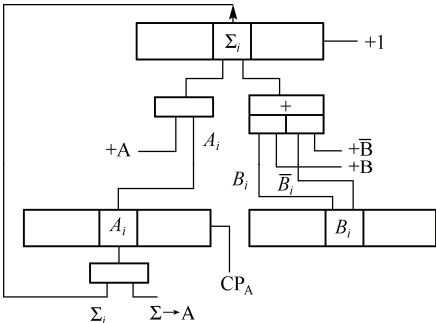


图 2-7 补码加减法运算器粗框

我们可以沿着“数据传送”这一线索去构成运算器，即分别考虑实现加法应当选取哪些操作数送入加法器，为此应当在有关的控制门设置哪些控制命令；实现减法又应当选取哪些操作数，设置哪些控制命令；一个运算器既能实现加法又能实现减法，则需使用与或逻辑形态将两种功能的控制命令综合起来。根据算法，作加法时应将 A、B 内容直接送加法器；作减法时将 A 内容送加法器，B 内容变补后送加法器；运算结果送回 A 寄存器。相应的控制命令列于表 2-5 中。

表 2-5 补码加减法所需控制命令

| 功能 | 所需控制命令 | | | | |
|----|--------|-------------|----|------------------------|--------|
| 加 | +A | +B | | $\Sigma \rightarrow A$ | CP_A |
| 减 | +A | $+ \bar{B}$ | +1 | $\Sigma \rightarrow A$ | CP_A |

控制命令的逻辑符号常以其逻辑含义的缩写形式来描述。例如，“+A”为1时（逻辑命题真），将A寄存器内容送入加法器相加。又如，“+ \bar{B} ”为1时，将B寄存器的反变量输出（ \bar{Q} 端，相当于B操作数变反）送加法器相加。“+1”表示令加法器初始进位为1，即末位加1。“ $\Sigma \rightarrow A$ ”表示将加法器输出按位送入A寄存器， CP_A 为其定时脉冲。上述控制命令将由控制器根据操作码产生，送至运算器。

如果进行加法运算，控制器将产生控制命令“+A”、“+B”、“ $\Sigma \rightarrow A$ ”、“ CP_A ”，即这几个命令为1，而“+ \bar{B} ”、“+1”等为0。这样便选择A与B进入加法器相加，结果送回A寄存器。A寄存器常被称为累加器。

如果进行减法运算，控制器将产生“+A”、“+ \bar{B} ”、“+1”、“ $\Sigma \rightarrow A$ ”、“ CP_A ”等，而“+B”为0。于是选择A与 \bar{B} 进入加法器相加，并且末位加1，这相当于 $A_{补} + (-B)_{补}$ ，即 $A_{补} - B_{补}$ 。

如果想获得更高的计算精度，可采取双倍字长运算，先做低位字运算，再做高位字运算。低位字运算的进位值可保留，作为高位字运算时的初始进位值，使两次运算能够衔接。

2.3.2 溢出的判断与移位

在计算机中，运算基本操作还包括溢出的判断、移位、舍入等。

1. 溢出判断方法

在加减运算中，当同号相加或异号相减时，运算结果的绝对值将增大。若该值超出了机器数的表示范围，则称为溢出。两个正数相加而绝对值超出允许的表示范围，称为正溢。两个负数相加而绝对值超出允许的表示范围，则称为负溢。一旦发生溢出，溢出的部分将被丢失，留下来的结果将不正确。如果只有一个符号位，溢出将使结果的符号位产生错乱。因此，计算机中应设置溢出判断逻辑，如果产生溢出，将停机并显示“溢出”标志；或者通过溢出处理程序，自动修改比例因子，然后重新运算。

我们先列举若干典型实例，其中有溢出与未溢出的情况。分析在什么情况下可能产生溢出，从中找到判断溢出的方法。设A、B两数字长5位，含1位数码。用 S_A 、 S_B 分别表示两数的数码，用 S_f 表示结果的符号，用 C_f 表示符号位产生的进位，用C表示最高有效数值位（即符号位右边的那位）产生的进位。

【例 2-37】 $9 + 3 = 12$

$$\begin{array}{r} 01001 \\ + 00011 \\ \hline 01100 \end{array} \quad (C_f = 0, C = 0)$$

【例 2-39】 $(-9) + (-3) = -12$

$$\begin{array}{r} 10111 \\ + 11101 \\ \hline 110100 \end{array} \quad (C_f = 1, C = 1)$$

【例 2-41】 $9 - 3 = 6$

$$\begin{array}{r} 01001 \\ + 11101 \\ \hline 100110 \end{array} \quad (C_f = 1, C = 1)$$

【例 2-38】 $11 + 7 = 18$ （正溢）

$$\begin{array}{r} 01011 \\ + 00111 \\ \hline 10010 \end{array} \quad (C_f = 0, C = 1)$$

【例 2-40】 $(-11) + (-7) = -18$ （负溢）

$$\begin{array}{r} 10101 \\ + 11001 \\ \hline 101110 \end{array} \quad (C_f = 1, C = 0)$$

【例 2-42】 $3 - 9 = -6$

$$\begin{array}{r} 00011 \\ + 10111 \\ \hline 11010 \end{array} \quad (C_f = 0, C = 0)$$

上述6例中，由于机器数的尾数为4位，补码表示，因此定点整数表示范围为： $-16 \sim +15$ ，超出这个范围即溢出（例2-38为正溢，例2-40为负溢）。在例2-39和例2-41中， C_f 表示超出模的部分，可舍去，但它不是溢出； $C=1$ ，也并未溢出。进位不等于溢出，因此不能简单地根据单

个进位信号去判断有无溢出，而应当从几个相关信号之间的关系去进行溢出判断。

先定义溢出标志 V ，若 $V=1$ 则表示有溢出， $V=0$ 则表示无溢出。

(1) 溢出判断逻辑一：

$$V = \bar{S}_A \bar{S}_B S_f + S_A S_B \bar{S}_f$$

这是从符号位之间的关系出发进行判断。第一项表明：两正数相加，即 $S_A=S_B=0$ ，若和为负数，即 $S_f=1$ （如例 2-38），则 $V=1$ 产生了正溢；第二项表明：两负数相加，即 $S_A=S_B=1$ ，若和为正数，即 $S_f=0$ （如例 2-40），则 $V=1$ 产生了负溢。注意，上式中第一项和第二项之间是“逻辑或”的关系，表明任何一项的值为逻辑 1，则必有溢出发生。

上述分析表明，只有同号数相加才可能产生溢出，而溢出的标志是结果数符 S_f 与两个操作数的数符刚好相反。

(2) 溢出判断逻辑二：

$$V = C_f \oplus C$$

这是从两种进位信号之间的关系出发判断溢出的， C_f 为符号位运算后产生的进位， C 为最高有效数位产生的进位。分析上述实例就会发现：产生正溢时，由于操作数较大，因而 $C=1$ ，但由于两个正数的符号位皆为 0，则 $C_f=0$ ；产生负溢时，由于补码映射值较小， $C=0$ ，但由于两个负数的符号位皆为 1，故 $C_f=1$ 。其他未溢出情况， C_f 与 C 皆相同。所以第二种判断逻辑可以理解成：当 C_f 与 C 不同时， $V=1$ 表明有溢出，相同时， $V=0$ 表明无溢出。这是在单符号位补码中应用较多的判别逻辑，尤其在无硬件乘除部件的低档计算机中。

(3) 溢出判断逻辑三

单符号位的信息量只能表示两种可能：数为正或为负，如果发生溢出，就会使符号位的含义产生混乱。将符号位扩充为两位，就能判别是否有溢出以及正确的结果符号。我们仍取前面的 4 个例子加以说明，操作数和结果均用双符号位表示。

| | |
|---|---|
| $9 + 3 = 12$ $\begin{array}{r} 001001 \\ + 000011 \\ \hline 001100 \end{array}$ $(-9) + (-3) = -12$ $\begin{array}{r} 110111 \\ + 111101 \\ \hline 110100 \end{array}$ | $11 + 7 = 18$ $\begin{array}{r} 001011 \\ + 000111 \\ \hline 010010 \end{array}$ $(-11) + (-7) = -18$ $\begin{array}{r} 110101 \\ + 111001 \\ \hline 101110 \end{array}$ |
|---|---|

从上述 4 例可定义双符号位的含义如下：00—结果为正，无溢出；01—结果正溢；10—结果负溢；11—结果为负，无溢出。

如果将第一符号位和第二符号位分别定义为 S_{f1} 和 S_{f2} ，则两个符号位不一致时表明有溢出。但不管结果是否有溢出，第一符号位 S_{f1} 将始终指示结果的正负性质。

溢出判断逻辑三：

$$V = S_{f1} \oplus S_{f2}$$

上式中，若 $S_{f1}=S_{f2}$ ，则 $V=0$ 无溢出；若 $S_{f1} \neq S_{f2}$ ，则 $V=1$ 有溢出。

采用多符号位的补码又叫变形补码，它的实质是扩大了模 M 。如果采用双符号位，则模 $M=4$ 。在乘除运算中广泛使用多符号位，但在主存中仍保持单符号位，运算时再扩充为多符号位，运算结束后又紧缩为单符号位，保存到主存之中。

2. 移位操作

移位是算术、逻辑运算的又一种基本操作，几乎所有机器指令系统中都设置有各类移位操作指令。通常将移位分为逻辑移位和算术移位两大类。

(1) 逻辑移位

在逻辑移位中，将数字代码当成纯逻辑代码，没有数值含义，因此没有符号与数值变化的概念。也可能是一组有数值含义的代码，但我们将它视为纯逻辑代码，通过逻辑移位对其进行判别、组装或某种加工。逻辑移位可分为：循环左移、循环右移、非循环左移、非循环右移等多种形式。

逻辑移位可应用在多种场合：利用移位操作将串行输入的数据组装成可并行输出的数据，即实现串→并转换；利用移位将并行输入的数据以串行方式输出，即实现并→串转换。为了对一串代码中的某一位进行判别（是 0 或 1？）或修改（置为 0、置为 1、变反修正），可以通过移位将该位左移至最高位，或右移至最低位，然后进行位判别或位修改，这样所花的硬件代价可以小些。

实现移位的硬件方法有两种。一是使用移位寄存器，在寄存器中实现移位，主机与外部进行串-并转换时常用这种方法。二是在寄存器间传送时利用斜位传送实现移位，如左斜一位相当于各位左移一位，在运算器内部常用这种方法实现移位，如乘除运算中。这里所说的移位逻辑，既适用于逻辑移位，也适用于算术移位。

(2) 算术移位

在算术移位中，数字代码具有数值含义，而且大多带有符号位。因此算术移位中必须保持符号位不变，一个正数在移位后应该还是正数。数值代码的各位有自己的权值，因此算术移位后将发生数值变化，如果移位前的二进制数为 X ，则左移一位变为 $2X$ ，右移一位后则变为 $X/2$ 。利用这一点，可以通过算术移位求一个数的 2^n 倍值。在算术移位中，正数补码和原码的尾数相同，因而它们的移位规则也相同，但正负数补码的移位规则却有区别。

① 正数补码（包括原码）移位规则：数符不变，空位补 0。

⊙ 左移——0.0101 左移一位成为 0.1010，00.1010 左移一位成为 01.0100。若采用单符号位，且移位前绝对值已大于等于 $1/2$ ，则左移后将溢出，因而不允许的。如果采用双符号位，模等于 4，则允许左移 1 位，第二符号位暂时用来保存有效数值，第一符号位仍能表示其数符，在除法运算中常常这样应用。

⊙ 右移——0.1010 右移一位成为 0.0101，01.0100 右移一位成为 00.1010。

注意：如果采用双符号位，则上次运算中暂存于第二符号位的数值，在右移后将移回至最高有效位，双符号位又恢复一致。在乘法运算中常有这种情况发生。

② 负数补码移位规则：数符不变，左移时空位补 0，右移时空位补 1。

⊙ 左移——1.1011 左移一位成为 1.0110，11.0110 左移一位成为 10.1100。若采用单符号位，且移位前绝对值已大于等于 $1/2$ ，即补码最高有效位为 0，则左移后将溢出，因而不允许的。如果采用双符号位，则允许左移 1 位，第二符号位暂时用来保存有效数值，第一符号位仍能表示其数符。

⊙ 右移——1.1011 右移一位成为 1.11011，1.0110 右移一位成为 1.10110，10.1100 右移一位成为 11.0110。由于补码负数的尾数与补码正数的尾数之间存在差别，即从最低位起，第一个 1 以后的各位应变反。这就不难理解上述右移规则：正数高位补 0，负数高位补 1。

3. 舍入处理

对于固定字长的数，右移将舍去低位部分。如两个 n 位字长的数相乘，可以产生 $2n$ 位乘积，但如果保持 n 位字长不变，则需将低位乘积舍去。又如，两个 n 位数相除，其商也可能超出 n 位，为限定字长也需舍去低位商。舍入的原则应该使本次舍入所造成的误差以及按相同舍入规则产生

的累计误差都比较小。下面介绍两种常用的简单舍入规则。

(1) 0 舍 1 入

原码与补码尾数的最末一位相同,因而采用相同的舍入方法。这种方法类似于十进制中的“四舍五入”。由于二进制中只有 0 与 1 之分,相应地采取“0 舍 1 入”。设舍入前是 $n+1$ 位,舍入后为 n 位,具体规则为:若第 $n+1$ 位是 0,则舍去后并不作修正;若第 $n+1$ 位是 1,则舍去后第 n 位加 1。

【例 2-43】 原码 0.1101, 舍入后得 0.111 (保留三位尾数);

原码 0.1100, 舍入后得 0.110。

【例 2-44】 补码 1.0011, 舍入后得 1.010;

补码 1.0100, 舍入后得 1.010。

舍入后产生了误差,但误差值小于末位的权值。

(2) 末位恒置 1

不管第 $n+1$ 位是 1 还是 0,舍去后将第 n 位恒置为 1。

【例 2-45】 原码 0.1101, 舍入后得 0.111 (保留三位尾数);

原码 0.1011, 舍入后得 0.101。

【例 2-46】 补码 1.0011, 舍入后得 1.001;

补码 1.0101, 舍入后得 1.011。

这种舍入方法比较简单,没有进位运算,在逻辑上易于实现。值得注意的是:当负数补码第 n 位被恒置 1 后,尾数的其余各位就与原码具有按位相反的简单对应关系,这在补码除法中求商值时很有用。

2.3.3 定点乘法运算

乘法运算比加减运算复杂。我们先举一个用手计算定点小数相乘的例子,这是大家熟悉的。如果将手算改为机器运算,将会遇到哪些问题?有哪些方法可以实现乘法运算?

【例 2-47】 $0.1101 \times 0.1011 = ?$

$$\begin{array}{r} 0.1101 \\ \times 0.1011 \\ \hline 1101 \\ 1101 \\ 0000 \\ 1101 \\ \hline 0.10001111 \end{array}$$

采用二进制后,乘法变得比较简单。对每一位乘数而言,无非要求乘以 1 或乘以 0,相应地让部分积累加被乘数或加 0。但由手算到机器实现,需要解决三个问题:① 符号问题;② 多项部分积相加,如何解决进位传递问题;③ 乘数权值每高一位,新部分积需左移一位,才能保持两次部分积之间的位权对应关系,这导致加法器位数增加,能否改变移位方法?

对于符号位的处理方法,可采用原码乘法或补码乘法。原码乘法是先取绝对值相乘,再根据同号相乘为正、异号相乘为负,单独决定乘积符号。补码乘法则让符号位直接参与乘法运算,算法将复杂一些。

如何处理多项部分积?这导致两种乘法器结构。一种乘法器是将 n 位乘法转换为 n 次累加及移位循环,因而可用常规加法器实现。即每次只处理一位乘数,每求得一项部分积,就立即累加

一次。相应地，将新部分积左移一位改为：原部分积的累加和右移一位。这样做的优点是：新旧部分积之间的位权对应关系不变，但原部分积中不再参与累加的低位已经右移，则加法器的位数不需扩充。这样，多位乘需分解为多步实现，依靠时序控制分步，所以又称为时序控制乘法器。如果计算机中只有常规的双操作数加法器，则软件实现乘法也采取类似的算法。以上是传统乘法器的实现途径，也可以每步处理两位乘数，使速度快些，称为两位乘法。

为了快速实现乘法运算，可以采取一种保存进位的策略，以解决多操作数直接相加时的进位结构问题，利用中、大规模集成电路芯片，在一拍中实现多项部分积的相加。这就形成了另一类乘法器结构，称为阵列乘法器。

本节主要讨论如何通过累加、移位实现分步乘法运算。

1. 原码一位乘法

原码一位乘法是指：取两操作数的绝对值（即原码中的尾数）相乘，每次将一位乘数所对应的部分积与原部分积的累加和进行相加，并右移一位。

(1) 分步乘法过程

【例 2-48】 $X = 0.1101$ ， $Y = -0.1011$ ，求 $XY = ?$

设寄存器 $A = 00.0000$ ， $B = |X| = 00.1101$ ， $C = |Y| = .1011$

| 步数 | 条件 | 操作 | A | C C_n ——判断位 |
|-----|-----------|----|-----------|---------------|
| | | | 00.0000 | .1011 |
| 第一步 | $C_n = 1$ | +B | + 00.1101 | |
| | | | 00.1101 | |
| | | → | 00.0110 | 1.1011 |
| 第二步 | $C_n = 1$ | +B | + 00.1101 | |
| | | | 01.0011 | |
| | | → | 00.1001 | 11.10 |
| 第三步 | $C_n = 0$ | +0 | + 00.0000 | |
| | | | 00.1001 | |
| | | → | 00.0100 | 111.1 |
| 第四步 | $C_n = 1$ | +B | + 00.1101 | |
| | | | 01.0001 | |
| | | → | 00.1000 | 1111 |

最后补加符号位，则乘积 $XY = 1.10001111$ 。

例 2-48 中的分步算式清晰地表明了计算过程、依据的条件、执行的操作及结果。采用这种规范化的描述形式可以避免许多错误。下面对分步运算过程进行分析。

① 寄存器分配与初始值

为了能用机器实现运算，需要设置相关的寄存器，用来存放有关的操作数与运算结果。在上例中，用 A 寄存器存放部分积的累加和，其初始值为 0；B 寄存器存放被乘数 X，在算式中只写其绝对值，即尾数部分，符号位另作处理；C 寄存器存放乘数 Y，在算式中也只写其绝对值，将符号位略去，以免计算时将符号位误作一位数字加以处理。C 寄存器的初始值是乘数 Y 的尾数，以后每乘一次，将已处理的低位乘数右移舍去，同时将 A 寄存器的末位移入 C 寄存器高位。乘法运算结束时，A 寄存器中存放着乘积的高位部分，C 寄存器中存放着乘积的低位部分。

② 符号位

A 和 B 均设置双符号位。因为部分积累加时最高有效数位有可能产生进位，如第二步结果，

这时可用第二符号位暂存该进位（它并非溢出），将来右移时把它移回尾数部分。

第一符号位始终指示部分积的符号，它决定了在右移时第一符号位补 0。如果只考虑原码一位乘，由于是先当成两个正数相乘，在最后加符号位前部分积肯定为正，所以也可将第一符号位略去，右移时符号位将始终补 0。但考虑到常将乘法器与除法器合成为一个乘除部件，除法运算需要双符号位，所以本例中采用双符号位，暂不考虑简化的可能。

③ 基本操作

在原码一位乘法中，每步只处理一位乘数，即位于 C 寄存器末位的乘数 C_n 。其余各位乘数将依次右移到该位，所以我们称之为判断位，它是决定每步操作的条件。

若 $C_n = 1$ ，执行 $A + B$ 操作，然后将累加和右移一位，用“ \rightarrow ”表示。

若 $C_n = 0$ ，执行 $A + 0$ 操作，然后右移。或直接让 A 右移一位。

右移时，A 的末位移入 C 的高位，A 的第二符号位移入尾数最高位，第一符号位移入第二符号位，而第一符号位本身则补 0。

④ 操作步骤

若乘数的尾数位数为 n （本例中 $n=4$ ），则需进行 n 次累加移位。机器实现时可用一个计数器来控制累加移位的操作循环次数。注意，本例中是先加后移位（包括可能的加 0 操作），最后一步累加后也必须右移一位。

（2）算法流程与逻辑实现

根据上述分析，可归纳出原码一位乘法的算法流程，如图 2-8 所示。其中 CR 是一个计数器，用来控制操作的步数。

现在来讨论原码一位乘法的硬件逻辑实现，即如何构成硬件乘法器。我们曾经强调过这样一种设计思路：即用控制数据传送的观点实现某种运算操作。因此，先拟出原码一位乘所需的各种传送控制命令，再根据这些命令设计加法器与各寄存器的输入逻辑。

原码一位乘的基本操作有两种： $A+B$ 或 $A+0$ 。相应地，需在加法器输入端设置 $+A$ 与 $+B$ 两个控制命令，以选择 A 与 B 进入加法器。如果只发 $+A$ 而不发 $+B$ ，则实现 $A+0$ 。在算式中，先将累加和送回 A 寄存器，再右移一位，机器实现时可以将两步合成一步，即将加法器输出右斜一位传回 A 寄存器，相应地将 A 寄存器输入门的控制命令用 $\Sigma/2 \rightarrow A$ 表示。此外，C 寄存器需右移，控制命令记作 \bar{C} 。A 寄存器与 C 寄存器都采用同步输入方式，需要同步打入脉冲 CP_A 、 CP_C 。原码一位乘所需的控制命令可归结如下：

$+A \quad +B \quad \Sigma/2 \rightarrow A \quad \bar{C} \quad CP_A \quad CP_C$

图 2-9 给出了一种原码一位乘的硬件逻辑框图。该逻辑图采取分段画法，即分别画出加法器、A、B、C 等寄存器。着重画出它们的输入逻辑，而加法器和寄存器本身却描述得较粗略。这种画法能更清晰地表现乘法器工作原理。

根据上述算法，如果乘数的尾数部分有 n 位，则分成 n 步实现 n 位乘。每一步所需操作时间称为一拍。在每一拍中，控制器根据乘法指令操作码与判断位 C_n ，发出有关控制命令，使乘法器实现一步操作。如果 $C_n=1$ ，则控制器发出 $+A$ 、 $+B$ 、 $\Sigma/2 \rightarrow A$ 、 \bar{C} 等微命令，实现 $A+B$ ，并将累加和右斜一位传回 A，同时将 C 右移一位。控制传送的电位型微命令将维持一拍时间，当运算结果稳定后，由同步脉冲 CP_A 与 CP_C 的上升沿将结果打入寄存器 A、C。如果 $C_n=0$ ，则控制器发 $+A$ 、 $\Sigma/2 \rightarrow A$ 、 \bar{C} 等微命令，实现 A 与 C 右移一位，由 CP_A 、 CP_C 打入。

乘积符号由异或逻辑形成，反相后置入 A 寄存器符号位 S_A 。由于累加与右移能在一拍中完成，不需要用第二符号位暂存进位信号，所以单纯的乘法器可以只设一个符号位。

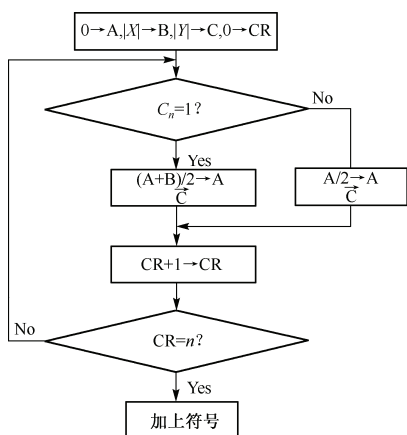


图 2-8 原码一位乘的算法流程

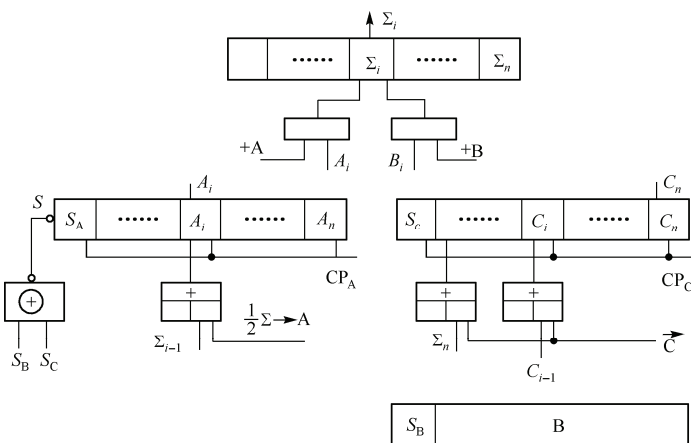


图 2-9 原码一位乘法器粗框图

寄存器 A 和 C 的输入端采用与或逻辑，一组输入门用来实现右移回传，另一组门没有细画，可用来输入操作数或作其他用途。

2. 补码一位乘法

原码乘法比较简单，但由于补码加减较原码加减简单，在通用计算机中常采用补码表示，从存储器中读得的操作数是补码表示的机器数。如果同一运算部件对加减运算采用补码算法，而对乘除运算又采用原码算法，就得进行码制转换，因而不方便，这就需要寻求补码乘法。补码乘法是指：操作数与结果均以补码表示，连同符号位一起，按相应算法运算。

实现补码乘法有两种方法。一种是先按原码乘法那样直接乘，再根据乘数符号进行校正。限于教材篇幅，对算法的证明从略，只简述其算法规则。

⊙ 不管被乘数 $X_{\text{补}}$ 的符号如何，只要乘数 $Y_{\text{补}}$ 为正，则可像原码乘法一样进行运算，其结果不需校正。

⊙ 如果乘数 $Y_{\text{补}}$ 为负，则先按原码乘法运算，结果再加一个校正量 $-X_{\text{补}}$ 。

这种算法称为校正法。

另一种方法是将校正法的两种情况统一起来，演变为比较法。由于它是 Booth 夫妇首先提出的，所以又称为 Booth 算法。Booth 算法是现在广泛采用的补码乘法，如果不加说明，一般的补码乘法就是指比较法。

(1) 比较法的算法分析

补码一位乘比较法可用下式表示：

$$[XY]_{\text{补}} = [X]_{\text{补}}[0.Y_1Y_2\cdots Y_n] - [X]_{\text{补}}Y_0 \quad (2-20)$$

式 (2-20) 概括了校正法的两种情况：若乘数为正，即 $Y_0=0$ ，则将 Y 的尾数乘以被乘数 $[X]_{\text{补}}$ ，不需校正；若乘数为负，即 $Y_0=1$ ，则 $[X]_{\text{补}}$ 乘以 $[Y]_{\text{补}}$ 的尾数后，再减去 $[X]_{\text{补}}$ 校正。

将式 (2-20) 改写为：

$$\begin{aligned} [XY]_{\text{补}} &= [X]_{\text{补}}[2^{-1}Y_1 + 2^{-2}Y_2 + \cdots + 2^{-n}Y_n] - [X]_{\text{补}}Y_0 \\ &= [X]_{\text{补}}[-Y_0 + (Y_1 - 2^{-1}Y_1) + (2^{-1}Y_2 - 2^{-2}Y_2) + \cdots + (2^{-(n-1)}Y_n - 2^{-n}Y_n) + 0] \\ &= [X]_{\text{补}}[(Y_1 - Y_0) + 2^{-1}(Y_2 - Y_1) + \cdots + 2^{-n}(0 - Y_n)] \\ &= [X]_{\text{补}}[(Y_1 - Y_0) + 2^{-1}(Y_2 - Y_1) + \cdots + 2^{-n}(Y_{n+1} - Y_n)] \end{aligned} \quad (2-21)$$

在机器实现中可在末位 Y_n 之后再增设一个附加位 Y_{n+1} ，其初始值为 0，对乘数 Y 的值并无影

响。若定义 $[A_0]_{\text{补}}$ 为初始部分积, $[A_1]_{\text{补}} \sim [A_n]_{\text{补}}$ 依次为各步求得的累加并右移后的部分积, 则可将式(2-21)改写为如下递推形式, 它更接近于乘法的分步运算形式。

$$\begin{aligned}
 [A_0]_{\text{补}} &= 0 \\
 [A_1]_{\text{补}} &= 2^{-1} \{ [A_0]_{\text{补}} + (Y_{n+1} - Y_n)[X]_{\text{补}} \} \\
 [A_2]_{\text{补}} &= 2^{-1} \{ [A_1]_{\text{补}} + (Y_n - Y_{n-1})[X]_{\text{补}} \} \\
 &\vdots \\
 [A_n]_{\text{补}} &= 2^{-1} \{ [A_{n-1}]_{\text{补}} + (Y_2 - Y_1)[X]_{\text{补}} \} \\
 [XY]_{\text{补}} &= [A_n]_{\text{补}} + (Y_1 - Y_0)[X]_{\text{补}}
 \end{aligned}
 \tag{2-22}$$

式(2-22)表明了补码一位乘的基本操作: 被乘数 $X_{\text{补}}$ 乘以对应的相邻两位乘数之差值, 再与原部分积累加, 然后右移一位, 形成该步的部分积的累加和。由于比较法是根据相邻两位乘数之差(低位减高位), 即两位的比较结果来决定相应操作, 所以称为比较法。因为每步要右移1位, 所以参与比较的两位始终是最末的两位, 即 Y_n 和 Y_{n+1} , 然后可根据这两位值决定相应操作, 其规则如表2-6所示。

表 2-6 补码一位乘的规则

| Y_n (高位) | Y_{n+1} (低位) | 操 作 |
|------------|----------------|-------------------------------------|
| 0 | 0 | $1/2 A_{\text{补}}$ |
| 0 | 1 | $1/2 (A_{\text{补}} + X_{\text{补}})$ |
| 1 | 0 | $1/2 (A_{\text{补}} - X_{\text{补}})$ |
| 1 | 1 | $1/2 A_{\text{补}}$ |

(2) 运算实例

【例 2-49】 $X = -0.1101$, $Y = -0.1011$, 求 $[XY]_{\text{补}} = ?$

设 $A = 00.0000$, $B = X_{\text{补}} = 11.0011$, $-B = -X_{\text{补}} = 00.1101$, $C = Y_{\text{补}} = 1.0101$ 。

| 步数 | 条件 | 操作 | A | C | C_{n+1} |
|-----|----|----|-----------|--------|-----------|
| 第一步 | 10 | | 00.0000 | 1.0101 | 0 |
| | | -B | + 00.1101 | | |
| | | → | 00.0110 | 11.010 | 1 |
| 第二步 | 01 | +B | + 11.0011 | | |
| 第三步 | 10 | | 11.1001 | 111.01 | 0 |
| | | -B | + 00.1101 | | |
| | | → | 00.0100 | 1111.0 | 1 |
| 第四步 | 01 | +B | + 11.0011 | | |
| 第五步 | 10 | | 11.0111 | 11111 | 0 |
| | | -B | + 00.1101 | | |
| | | | 00.1000 | 1111 | |

则 $[XY]_{\text{补}} = 0.1000111$ 。

本例以分步算式的形式表明了补码一位乘法的运算过程, 下面再对有关要点进行说明。

① 初始值和符号位

A 寄存器用来存放部分积的累加和, 初始值为 0, 采取双符号位。累加时可能产生的进位暂存于第二符号位, 第一符号位始终指示部分积的累加和的正负, 以控制右移时补 0 或补 1。补码乘法中有加或减, 相应地, 部分积可能有正有负, 这就需要第一符号位保持正负标志。

B 寄存器中存放补码表示的被乘数 $X_{\text{补}}$, 双符号位 (与 A 相对应)。补码一位乘的基本操作有 $A+B$ 和 $A-B$, 所以算式中先写出 $X_{\text{补}}(B)$ 和 $-X_{\text{补}}(-B)$ 的值。

C 寄存器存放补码表示的乘数 $Y_{\text{补}}$ 。取单符号位, 以控制最后一步操作, 该步操作体现了乘数正负的影响。 Y 的末位添 0, 称为附加位 $Y_{n+1}(C_{n+1})$ 。

② 基本操作

用 C 寄存器最末两位 (含增加的 C_{n+1}) 作判断位, 按前述规则决定各步操作。如第①步, 判断位为 10, 低位减高位得 -1, 所以执行 $A-B$, 然后右移一位。又如第②步, 判断位为 01, 低位减高位得 1, 执行 $A+B$ 。

每步加/减后让 A 与 C 右移一位, 而每次是相邻两位乘数比较, 所以除去 C 的符号位和增加的末位 0 以外, 各乘数位将参加两次比较。注意, 现在讨论的是补码一位乘, 比较两位只相当于处理一位乘数, 不可误认为两位乘。

③ 移位

在右移时, 第二符号位值移入尾数的最高数位, 第一符号位值移入第二符号位, 第一符号位本身不变, 而 A 寄存器末位移入 C 寄存器。

④ 步数与最后一步操作

如上例, 乘数的有效尾数是 4 位, 共做 5 步。除 C_0 、 C_{n+1} 位外, 有效尾数中各位均参与 2 次比较。注意, 最后一步不移位, 因为这一步是用来处理符号位的, 按式 (2-22), 这一步不再移位。

(3) 逻辑实现

根据补码一位乘法构成乘法器, 其设计方法与原码乘法器的设计相似。即先拟定各种操作所需的控制命令, 再据此确定加法器与各寄存器的输入逻辑。其基本思路仍是围绕着数据传送这一线索。所需的控制命令有:

$$+A \quad +B \quad +\bar{B} \quad +1 \quad \Sigma \rightarrow A \quad \Sigma/2 \rightarrow A \quad \bar{C} \quad CP_A \quad CP_C$$

相应地, 加法器 A 输入端需设置一个控制门, 由 +A 控制选择 A 寄存器数据。加法器 B 输入端需设置一个与或门, 由 +B 控制选择 B 寄存器内容; 由 + \bar{B} 控制选择 B 寄存器的反码, 同时在末位加 1, 即让初始进位为 1。A 寄存器输入需设置与或门, 其中一组由 $\Sigma/2 \rightarrow A$ 控制, 使加法器输出右斜一位传回寄存器 A。C 寄存器输入中的一组可实现 C 右移功能。 CP_A 和 CP_C 分别为 A、C 寄存器的定时打入脉冲。上述控制命令由控制器分时发出, 根据 $C_n C_{n+1}$ 比较结果发 +B 或 + \bar{B} 与 +1。

我们不再细画硬件框图, 读者可根据上述设计方法自行构成补码乘法器。

3. 原码两位乘法

在使用常规双操作数加法器的前提下, 如何提高乘法速度呢? 一种合乎逻辑的途径是每步同时处理两位乘数, 根据两位乘数的组合决定本步应该做什么操作, 从而在一步内求得与两位乘数相对应的部分积, 称为两位乘法。其运算速度将比一位乘提高近 1 倍。

(1) 算法分析

若两位乘数按高低顺序为 $Y_i Y_{i+1}$, 则 4 种可能组合所对应的操作如下:

| Y_i | Y_{i+1} | 操 作 |
|-------|-----------|------------------|
| 0 | 0 | 原部分积右移 2 位 |
| 0 | 1 | 原部分积+X, 再右移 2 位 |
| 1 | 0 | 原部分积+2X, 再右移 2 位 |
| 1 | 1 | 原部分积+3X, 再右移 2 位 |

以上是原码两位乘本应执行的操作, 但机器实现时有困难。因为加法器可实现的操作有+X和-X, 通过将加数左斜一位送加法器可实现+2X, 但+3X如何实现? 这就需要做出转换。可将+3X当成4X-X来处理, 即本步只执行-X, 用一个欠账触发器 C_j 记下欠账, 到下一步再补上+4X。注意, 每一步累加后部分积要右移两位, 移位前后相差4倍, 所以前一步移位前欠下的+4X操作, 到了移位后的下一步, 只需执行+X操作即可。不难通过实例证明这一相对关系。

由此可得到, 实用的原码两位乘法规则如表 2-7

表 2-7 原码两位乘的规则

| Y_i | Y_{i+1} | C_j | 操 作 | |
|-------|-----------|-------|--------------|---------------------|
| 0 | 0 | 0 | $A/4$ | $0 \rightarrow C_j$ |
| 0 | 0 | 1 | $1/4 (A+X)$ | $0 \rightarrow C_j$ |
| 0 | 1 | 0 | $1/4 (A+X)$ | $0 \rightarrow C_j$ |
| 0 | 1 | 1 | $1/4 (A+2X)$ | $0 \rightarrow C_j$ |
| 1 | 0 | 0 | $1/4 (A+2X)$ | $0 \rightarrow C_j$ |
| 1 | 0 | 1 | $1/4 (A-X)$ | $1 \rightarrow C_j$ |
| 1 | 1 | 0 | $1/4 (A-X)$ | $1 \rightarrow C_j$ |
| 1 | 1 | 1 | $A/4$ | $1 \rightarrow C_j$ |

所示。该表的左栏是判断条件, 其中含两位乘数, 另有一位欠账触发器 C_j 。注意, C_j 不是一位乘数。 Y_i 、 Y_{i+1} 、 C_j 组成三位判断位。表的右栏是应执行的基本操作, A 表示部分积, $A/4$ 表示将 A 右移两位。例如第 4 行, $Y_i Y_{i+1} = 01$, 本应执行+X; 但 $C_j = 1$, 表明上一步欠账+4X, 本步应补还+X; 所以综合结果应执行+2X, 不再欠账, 因而 C_j 变为 0。又如第 6 行, $Y_i Y_{i+1} = 10$, 本应执行+2X; 但 $C_j = 1$, 表明欠账+4X, 本步应补还+X; 综合结果应执行+3X; 实际改作-X, 记下欠账+4X, 所以 C_j 变为 1。

(2) 运算实例

【例 2-50】 $X = -0.111111$, $Y = 0.111001$, 求 $XY = ?$

令 $A = 000.000000$, $B = |X| = 000.111111$, 则 $-B = 111.000001$, $+2B = 001.111110$, $C = |Y| = 00.111001$ 。

| 步数 | 条件 | 操作 | A | C | C_j |
|-----|-----------------|-----------------|--------------|-----------|-------|
| | $C_{n-1}C_nC_j$ | | 000.000000 | 00.111001 | 0 |
| 第一步 | 0 1 0 | +B | + 000.111111 | | |
| | | | 000.111111 | | |
| | | $\rightarrow 2$ | 000.001111 | 1100.1110 | 0 |
| 第二步 | 1 0 0 | +2B | + 001.111110 | | |
| | | | 010.001101 | | |
| | | $\rightarrow 2$ | 000.100011 | 011100.11 | 0 |
| 第三步 | 1 1 0 | -B | + 111.000001 | | |
| | | | 111.100100 | | |
| | | $\rightarrow 2$ | 111.111001 | 00011100. | 1 |
| 第四步 | 0 0 1 | +B | + 000.111111 | | |
| | | (还账) | 000.111000 | 000111 | |

加符号位, 则乘积 $XY = 1.111000000111$ 。

对运算过程的几点说明如下。

① 初始值与符号位。根据原码乘法规则, 先取绝对值相乘, 用 B 寄存器存放被乘数 X 的绝对值, C 中存放乘数 Y 的绝对值。可能的操作有+B、-B、+2B, 所以在算式中先写出 B 的机器负数与 2B, 机器实现时是通过 + \bar{B} 与末位+1 实现-B (将 B 变补相加), 通过将 B 左斜一位送加法器

实现+2B。

A 与 B 均取三符号位。因为有+2B 操作，2B 本身有可能进位到第三符号位，累加后则可能进位到第二符号位，如本例中第二步所示。操作中可能有-B，部分积有可能暂时为负（补码表示），需由第一符号位指示正负，以控制移位操作，如第三步。

C 取双符号位 00，在最后一步还清欠账时，双符号位就充当判断位中的乘数，如第四步。注意，每步处理两位乘数，所以应使乘数的尾数保持为偶数位；如果不是，就应在末位后补 0，以凑足偶数位。

② 欠账触发器 C_j 。与补码乘中的附加位 C_{n+1} 不同， C_j 是一位独立的触发器，不是乘数的一部分。运算前没有欠账， C_j 初始值为 0。以后每一步根据欠不欠账来决定 C_j 的状态，因此 C_j 的值并不是由移位获得的。

③ 操作。依据表 2-7 所示的规则决定操作，“→2”表示右移 2 位。

④ 步数。乘数的尾数有 n 位，需要做 $n/2$ 步操作。如果最后一步欠了账，还需增加一步还账，如本例，共做 4 步。注意，最后一步不移位。

（3）逻辑实现

按上述分析来设计原码两位乘法器，可拟出各种数据传送所需的控制命令如下：+A、+B、+2B、+ \bar{B} 、+1、0→ C_j 、1→ C_j 、 $\Sigma \rightarrow A$ 、 $\Sigma/4 \rightarrow A$ 、 $C/4 \rightarrow C$ 、 CP_A 、 CP_C 。

相应地，加法器 A 输入端需设置一个控制门，由+A 控制；加法器 B 输入端需设置一个与或门，视操作需要选择+B、+2B、-B；对 C_j 应有复位、置 1 功能；由 $\Sigma/4 \rightarrow A$ 控制将加法器输出右斜两位传回寄存器 A； $C/4 \rightarrow C$ 控制 C 寄存器每步右移两位； CP_A 与 CP_C 为同步定时脉冲。根据以上命令和相应逻辑可以构成原码两位乘法器。

关于补码两位乘法规则，可结合原码两位乘的思想与补码一位乘比较法进行推导，这里不再详述。

2.3.4 定点除法运算

用手算方式做二进制除法运算时，所遵循的规则是：比较被除数与除数的绝对值大小，够减（被除数绝对值大于除数绝对值）商 1，用被除数减去除数，将余数左移（末位补 0），再与除数比较，以便求下一位商；不够减（被除数绝对值小于除数绝对值）商 0，不做减法，将余数左移，继续与除数比较。可将被除数看作初始余数。因此，除法运算的关键是比较余数与除数的大小，即判断是否够减。当将手算方法转变为机器实现时，如何判断够减？如何处理符号位？又如何提高除法运算速度呢？

① 如何判断够减

一种方法是先用逻辑电路进行比较判别：如果够减才执行减法，并商 1；如果不够减，就不再进行减操作，商 0。这种方法增加了硬件代价，又无明显的优点，因而很少采用。

另一种方法是用减法试探，即在减后根据余数与除数的符号比较，判断本次减操作究竟够减还是不够减。如果判明不够减，怎么办？这又派生出两种算法，其一是恢复余数法，其二是恢复余数法。

恢复余数法的处理思想是：先减后判，如果减后发现不够减，则商 0，并加除数，恢复减前的余数，相当于取消这一步已做的减操作，再往后运算。这是一种基于除法基本算法的处理方法，既增加了一些不必要的操作，又使操作步数随着不够减情况出现的次数而变化。操作步数不固定将给控制时序的安排带来一些困难，并增加了运算时间，因而也已很少采用。

不恢复余数除法又称为加减交替除法，其处理思想是：先减后判，如果减后发现不够减，则在下一步改做加除数操作。这样，操作步数是固定的，仅与商的位数有关，因此这是现在普遍采用的算法。

② 如何处理符号位

与乘法相似，除法也分为原码除法与补码除法两类。原码除法是先取绝对值相除（即取原码的尾数），符号位单独处理：同号相除为正、异号相除为负。补码除法是带符号的补码直接相除，这就带来一个问题：补码商有正有负，且正商、负商的尾数是不同的，如何确定上商的规则？

③ 如何提高除法运算速度

与乘法相似，常见除法器有以下 3 种。

- ⊙ 传统的除法可以利用常规的双操作数加法器，将除法分解为若干次“加减与移位”循环，由时序控制分步实现。
- ⊙ 为了提高速度，一种途径是采用迭代除法，将除法转换为乘法处理，就可以利用快速乘法器实现除法。
- ⊙ 阵列除法器，一次求得商和余数，这已成为实现快速除法的基本途径。
- ⊙ 本节主要讨论第一种方法，即用加减与移位分步实现的不恢复余数除法。

1. 原码不恢复余数除法

(1) 算法分析

为了推导不恢复余数算法，我们先讨论恢复余数的情况。

设 Y 表示除数， r 表示余数。第 i 步将余数左移一位后减除数 ($2r_{i-1} - Y$)，则其上商与下一步操作可能出现两种情况：若够减，即余数 $r_i = 2r_{i-1} - Y > 0$ ，则商 $Q_i = 1$ ，下一步做 $r_{i+1} = 2r_i - Y$ ；若不够减，即 $r'_i = 2r_{i-1} - Y < 0$ （由于不够减，暂将余数记为 r'_i ，有别于恢复后的余数 r_i ），则商 $Q_i = 0$ ，恢复余数为 $r_i = r'_i + Y = 2r_{i-1}$ ，下一步做 $r_{i+1} = 2r_i - Y$ 。这里：

$$\begin{aligned} r_{i+1} &= 2r_i - Y = 2(r'_i + Y) - Y \\ &= 2r'_i + Y \end{aligned} \quad (2-23)$$

式(2-23)表明：当出现不够减情况时，也可以不恢复余数而直接做下一步，将操作改为 $2r'_i + Y$ ，其结果与恢复余数后再减 Y 是等效的。这就是不恢复余数除法，也称为加减交替法。

原码不恢复余数除法的要点如下：

- ① 取绝对值（原码尾数）相除，符号位单独处理。
- ② 对于定点小数除法，为使商不致溢出，要求被除数绝对值小于除数绝对值，即 $|X| < |Y|$ 。
- ③ 每步操作后，可根据余数 r_i 符号判断是否够减： r_i 为正，表明够减，上商 $Q_i = 1$ ； r_i 为负，表明不够减，上商 $Q_i = 0$ 。
- ④ 基本操作可用通式描述为

$$r_{i+1} = 2r_i + (1 - 2Q_i)Y \quad (2-24)$$

式(2-24)包含了两种情况：若第 i 步够减， $Q_i = 1$ ，则第 $i+1$ 步应做 $2r_i - Y$ ；若第 i 步不够减， $Q_i = 0$ ，则第 $i+1$ 步应做 $2r_i + Y$ 。

- ⑤ 原码除的思想是先当成正数相除，若最后一步所得余数为负，则恢复余数，以保持 $r \geq 0$ 。

(2) 运算实例

【例 2-51】 $X \div Y = -0.10110 \div 0.11111 = ?$

设 $A = |X| = 00.10110$ ， $B = |Y| = 00.11111$ ，则 $-B = 11.00001$ ， $C = |Q| = 0.00000$ 。

| 步数 | 条件 | 操作 | A | C | C_n |
|-----|-----------|------|-------------------|--------|---------------|
| | | | 00.10110 | r_0 | 0.00000 |
| 第一步 | | ← | 01.01100 | $2r_0$ | |
| | | -B | <u>+ 11.00001</u> | | |
| | $S_A = 0$ | | 00.01101 | r_1 | 0.00001 Q_1 |
| 第二步 | $C_n = 1$ | ← | 00.11010 | $2r_1$ | |
| | | -B | <u>+ 11.00001</u> | | |
| | $S_A = 1$ | | 11.11011 | r_2 | 0.00010 Q_2 |
| 第三步 | $C_n = 0$ | ← | 11.10110 | $2r_2$ | |
| | | +B | <u>+ 00.11111</u> | | |
| | $S_A = 0$ | | 00.10101 | r_3 | 0.00101 Q_3 |
| 第四步 | $C_n = 1$ | ← | 01.01010 | $2r_3$ | |
| | | -B | <u>+ 11.00001</u> | | |
| | $S_A = 0$ | | 00.01011 | r_4 | 0.01011 Q_4 |
| 第五步 | $C_n = 1$ | ← | 00.10110 | $2r_4$ | |
| | | -B | <u>+ 11.00001</u> | | |
| | $S_A = 1$ | | 11.10111 | r'_5 | 0.10110 Q_5 |
| 第六步 | $C_n = 0$ | +B | <u>+ 00.11111</u> | | |
| | | 恢复余数 | 00.10110 | r_5 | |

则商 = -0.10110, 余数 = -0.10110 × 2⁻⁵。对运算过程的说明如下:

① 寄存器分配与符号位。A 寄存器中开始存放被除数的绝对值, 以后将存放各次余数。A 取双符号位, 左移一位时, 有效数位可能需暂时存放在第二符号位; 第一符号位指示正负, 可判断是否够减, 从而决定商值。

B 寄存器存放除数的绝对值, 取双符号位与 A 相对应。由于有 ±B 两种操作, 在算式中我们将 -B 值也写出。

C 寄存器用来存放商的绝对值, 取单符号位。在机器实现中商由末位置入, 在每次置入新商的同时, 原有的商同时左移一位。

② 第一步操作。将被除数 X 视为初始余数 r_0 , 在 $|X| < |Y|$ 的前提下, 第一步操作为 $2r_0 - Y$ 。

③ 基本操作和上商。如前所述, 基本操作依据通式 $r_{i+1} = 2r_i + (1 - 2Q_i)Y$, 体现了加减交替法思想。而商值则根据各步余数 r_i 的符号来决定, r_i 为正, 上商 1; r_i 为负, 上商 0。

按移位规则, 左移时末位补 0, 尾数的最高位将移入第二符号位暂存。

④ 操作步数与最后一步操作。如果要求 n 位商 (不含符号位), 则需做 n 步“左移、加减”循环; 若第 n 步余数 r_n 为负, 则需增加一步恢复余数, 使最终的余数仍为绝对值形式。增加的这一步不移位。如本例, 5 步求得 5 位商, 再增加一步恢复余数。

⑤ 结果表达: 除法的结果是求得商及余数。余数左移后, 在形式上提高了位权, 因此余数的实际位权要比形式上书写的低, r_n 的位权应乘以 2^{-n} , 如本例中最后余数 r_5 的实际值是 0.10110×2^{-5} 。

原码除是先当成正数相除, 求得商与余数的绝对值后, 商符按同号相除为正, 异号相除为负确定; 余数的实际符号与被除数的符号相同。

(3) 逻辑实现

在逻辑实现中, 可将左移一位与加减合为一步进行, 相应地需设置下列微命令:

| | |
|------------------------|--------------------|
| +2A、+A | 加法器 A 输入端选择命令 |
| +B、+ \bar{B} 、+1 | 加法器 B 输入端选择命令及初始进位 |
| $\Sigma \rightarrow A$ | A 寄存器输入选择命令 |

| | |
|-----------------------|-------------|
| \bar{C} | C 寄存器左移控制命令 |
| $Q_i \rightarrow C_n$ | 置入商值 Q_i |
| $CP_A、CP_C$ | 同步打入脉冲 |

根据以上命令可设计加法器、A 寄存器、C 寄存器的输入逻辑，从而组成除法器。

2. 补码不恢复余数除法

补码除法是指被除数、除数、所求得商、余数等都补码表示。由于符号位要参与运算，与原码除法的绝对值运算相比，补码除法需要解决一些新的问题，即如何根据操作数的符号决定实际操作？如何判断够减？如何求商值？如何确定商符？由于数符可正可负，这些问题的解决要比原码除法相对复杂一些。

表 2-8 概括了补码不恢复余数法（也称为加减交替法）的运算规则，我们对其中的关键问题进行分析和讨论。

表 2-8 补码除法规则

| $X_{\text{补}} Y_{\text{补}}$ 数符 | 商符 | 第一步操作 | $r_{\text{补}} Y_{\text{补}}$ 数符 | 上商 | 下一步操作 |
|--------------------------------|----|-------|--------------------------------|-----|---|
| 同号 | 0 | 减 | 同号（够减） 异号（不够减） | 1 0 | $2[r_i]_{\text{补}} - Y_{\text{补}}$ $2[r_i]_{\text{补}} + Y_{\text{补}}$ |
| 异号 | 1 | 加 | 同号（不够减） 异号（够减） | 1 0 | $2[r_i]_{\text{补}} - Y_{\text{补}}$ $2[r_i]_{\text{补}} + Y_{\text{补}}$ |

（1）判断够减

随着除法运算的进行，应使余数的绝对值越除越小。因此，被除数与除数同号时，两数应当相减；两数异号时，则应相加。这是带符号数相除的特点。下面通过两组例子说明操作与数符之间的关系，以便找出判断够减的方法（为便于理解，以带符号的十进制真值为例）。

【例 2-52】 同号相除： ① $7 \div 4$ 1 ② $4 \div 7$ 0

| | | |
|---|---|--|
| $\begin{array}{r} 1 \\ 4 \overline{) 7} \\ \underline{-4} \\ 3 \end{array}$ | $\text{---}X$ $\text{---}Y$ $\text{---}r$ | $\begin{array}{r} 0 \\ 7 \overline{) 4} \\ \underline{-7} \\ -3 \end{array}$ |
| 够减 | | 不够减 |

【例 2-53】 异号相除： ① $-7 \div 4$ 1 ② $-4 \div 7$ 0

| | | |
|---|---|---|
| $\begin{array}{r} 1 \\ 4 \overline{) -7} \\ \underline{+4} \\ -3 \end{array}$ | $\text{---}X$ $\text{---}Y$ $\text{---}r$ | $\begin{array}{r} 0 \\ 7 \overline{) -4} \\ \underline{+7} \\ +3 \end{array}$ |
| 够减 | | 不够减 |

从以上例子可以看出，用余数 r 与被除数 X 进行数符比较可以判断是否够减，如例 2-52 与例 2-53 的①， r 、 X 同号表明够减；而两组例子中的②， r 、 X 异号表明不够减。但运算后 X 将被新的余数 r 所取代，不再保留，而除数 Y 则在运算中一直保持不变的，所以我们改用 Y 与 r 作数符比较。在同号相除时，若 r 、 Y 同号表明够减；若 r 、 Y 异号表明不够减。在异号相除时，若 r 、 Y 同号反而表明不够减；若 r 、 Y 异号则表明够减。

（2）求商值

同号相除商为正，够减商 1，不够减商 0。

异号相除商为负，怎样上商？若对补码商采取“末位恒置 1”的舍入方法，则负数补码与正数之间将变为一种简单的对应关系，即除去恒定为 1 的末位外，尾数的其他各位与正数相反。因此当商为负数补码时，够减商 0，不够减商 1。

结合对够减与否的判别方法，恰巧使求商值的规则在形式上统一起来，如表 2-8 所示。即不管够减或不够减，只要 r_i 、 Y 同号便商 1，异号便商 0。因此，商值逻辑为：

$$Q_i = \overline{S_{r_i} \oplus S_Y} \quad (2-25)$$

式中， S_{r_i} 为余数 r_i 的数符， S_Y 为除数 Y 的数符。

(3) 确定商符

补码符号位参加运算，因而商符是通过运算求得的，在补码除法中有两种常见做法。不管采用哪种做法，最后的商符应与实际商符一致，即同号相除商符为 0，异号相除商符为 1。

第一种做法：先做 $X_{\text{补}} \pm Y_{\text{补}}$ ，按表 2-8 中求商值的规律求商符。如果 X 、 Y 同号，做 $X_{\text{补}} - Y_{\text{补}}$ ，由于定点小数除法要求 $|X| < |Y|$ ，所以第一步操作结果肯定是不够减的， r_i 、 Y 异号，商符为 0。如果 X 、 Y 异号，做 $X_{\text{补}} + Y_{\text{补}}$ ，由于不够减， r_i 、 Y 同号，商符为 1。

这种方法的优点是：求商符与求商值的规则一致。缺点是：第一步做 $X_{\text{补}} \pm Y_{\text{补}}$ ，以后各步做 $2r_{\text{补}} \pm Y_{\text{补}}$ ，操作不统一，控制上稍为复杂一些。

第二种做法：一开始就将被除数 X 当成初始余数 r_0 ， r_0 、 Y 同号时商符为 1， r_0 、 Y 异号时商符为 0。然后根据商符第一步做 $2r_{0\text{补}} \pm Y_{\text{补}}$ 。显然，按商值规律求得的商符与实际商符相反，将它称作假商符，通过求反将它校正。

这种方法将求商符与求商值的规则先暂时统一起来，各步操作规律也是统一的。虽然需对商符进行校正，但这并不困难。因此我们推荐第二种做法。

(4) 对商校正

如果采用第二种做法，就存在一个商的校正问题，包括将商的末位恒置 1 和将商符变反。我们略去严格的算法推导，仅从概念上简明地说明校正方法。

如果需求 n 位商（不含符号位），则做 n 步，求得假商符与 $n-1$ 位商，称为假商，然后令假商加 $1+2^{-n}$ ，即获得校正后的真商。假商符加 1 并舍去进位，就变反为真商符；尾数加 2^{-n} 相当于令第 n 位商恒为 1，符合前面推出求商规则时所用的前提（末位恒置 1）。对求得的商，误差控制在末位。

【例 2-54】 $X \div Y = 0.1000 \div (-0.1010) = ?$

设 $A = X_{\text{补}} = 00.1000$ ， $B = Y_{\text{补}} = 11.0110$ ， $-B = 00.1010$ ， $C = Q_{\text{补}} = 0.0000$ 。

| 步数 | 条件 | 操作 | A | C | C_{n-1} |
|-----|----------------|-----|-----------|--------|-------------|
| 第一步 | r_0 、 Y 异号 | 求商符 | 00.1000 | r_0 | 0.000 Q_0 |
| | $C_{n-1} = 0$ | ← | 01.0000 | $2r_0$ | |
| | | +B | + 11.0110 | | |
| 第二步 | r_1 、 Y 异号 | | 00.0110 | r_1 | 0.000 Q_1 |
| | $C_{n-1} = 0$ | ← | 00.1100 | $2r_1$ | |
| | | +B | + 11.0110 | | |
| 第三步 | r_2 、 Y 异号 | | 00.0010 | r_2 | 0.000 Q_2 |
| | $C_{n-1} = 0$ | ← | 00.0100 | $2r_2$ | |
| | | +B | + 11.0110 | | |
| 第四步 | r_3 、 Y 同号 | | 11.1010 | r_3 | 0.001 Q_3 |
| | $C_{n-1} = 0$ | ← | 11.0100 | $2r_3$ | |
| | | -B | + 00.1010 | | |
| | | | 11.1110 | r_4 | |

则假商 = 0.001，真商 = $0.001 + 1.0001 = 1.0011_{\text{补}} = -0.1101_{\text{真值}}$ ；

余数 $=2^{-4}r_4=1.11111110_{\text{补}}=-2^{-4}\times 0.0010_{\text{真值}}$ 。

相关说明如下：

① 寄存器分配与符号位。用 A 寄存器存放被除数（补码），以后存放余数，取双符号位。B 寄存器存放除数（补码），双符号位，在算式中先写明 $-B$ 。C 寄存器存放商，初始值为 0（未考虑商符之前），单符号位。

② 假商符。在第一步操作之前，先根据 r_0 、 Y 符号比较确定假商符（与真商符相反）。

③ 基本操作。各步操作统一。根据假商符值决定第一步操作，并根据第一步操作结果决定第一位商值。如本例，假商符 Q_0 为 0，第一步做 $2r_0+B$ ，然后根据 r_1 、 Y 异号上商 Q_1 为 0。

④ 步数。本例求 4 位商（尾数），所以做 4 步，但假商中只取 3 位商，第 4 位商通过校正（恒置 1）获得。

⑤ 假商校正。本例中假商为 0.001，含一位假商符与三位商值。校正（加 1.0001）后得到真商 1.0011，商符由 0 校正为实际值 1，末位商恒为 1。

⑥ 结果表达。补码相除后，商与余数自带符号（算式中最后一步求得的余数符号就是实际的余数符号），可正可负，可以分别写出商与余数的补码或真值形式。注意余数的权。

逻辑实现中所需的微命令与原码除法微命令相同，因此补码除法器在逻辑结构上与原码除法器基本相同，区别仅在于微命令形成逻辑与商值形成逻辑是根据补码除法规则产生的。

2.3.5 浮点四则运算

浮点数比定点数的表示范围宽，有效精度高，更适合于科学与工程计算的需要。当要求计算精度较高时，往往采用浮点运算。但浮点数的格式较定点数格式复杂，硬件实现的成本较高，完成一次浮点四则运算所需的时间也比定点运算长。因此，计算机按其运算功能分为几种档次，较低档的计算机在硬件上只有定点运算功能，通过软件子程序实现浮点运算；一些微型计算机的 CPU 没有硬件浮点运算功能，但另有配套的浮点处理器，或称为协处理器，可将有关的浮点运算送往协处理器处理，以提高运行速度；功能较强的计算机配置有专门的浮点运算部件，相应地，指令系统中包含浮点运算指令。

浮点代码中包含两组定点代码：采用定点整数格式的阶码和采用定点小数格式的尾数，另隐含约定阶码的底 R 。因此，浮点运算实质上包含两组定点运算：阶码运算和尾数运算。但这两部分有各自的作用和相互间的关联。

如前所述，浮点数有规格化的和非规格化的。相应地，浮点运算也可分为两类，有的计算机只能执行其中的一种，有的则允许选择。规格化浮点运算的有效精度较高，使用较多。下面主要讨论规格化浮点运算，并强调对阶和规格化两个基本概念。

1. 浮点加减运算

规格化浮点加减运算可按以下步骤进行。

（1）检测能否简化操作

浮点运算较为复杂，能简化则尽量简化，一个最简单的方法就是判断操作数是否为零。当加数（减数）为零时，运算结果就是被加数（被减数）。当被加数（被减数）为零时，运算结果就是加数（减数）。当被加数（被减数）为零时，运算结果就等于减数变补。因此，当操作数之一为零时，就可以简化操作，免去不必要的后续对阶操作。

如何判断浮点数为 0？一种情况是尾数为 0，另一种情况是阶码下溢作为机器零处理。如果

是 IEEE 754 格式的浮点数, 则当阶码和尾数均为 0 时, 浮点数才为 0 值。

(2) 对阶

在浮点数中, 尾数是定点小数, 而阶码体现出放大或缩小的比例因子。只有阶码相同的数, 其尾数的真正权值才相同, 才能让尾数直接加减。因此, 两浮点数加减时, 必须将它们的阶码调整得一样大, 这个过程称为对阶, 这是浮点加减中关键的一步。

$$\begin{array}{ccc} \text{【例 2-55】} & 2^3 \times 0.100100 & \\ & \xrightarrow{\text{对阶}} & 2^3 \times 0.100100 \\ & 2^1 \times 0.110100 & 2^3 \times 0.001101 \end{array}$$

上例中, 被加数的阶码为 3, 加数的阶码为 1, 因此它们的尾数不能直接相加。只有通过将对阶将阶码调整到相同后, 才能让尾数相加。显然, 对阶的规则是: 阶码小的数向阶码大的数对齐。换句话说, 以大的阶码为基准, 调整小的阶码。

当调整阶码时, 尾数应同步地移位, 以保持浮点数的值不变。如果阶码以 2 为底, 则每当阶码增 1 时尾数应右移 1 位, 有可能舍去低位, 但将保留有效的高位部分, 总值基本不变。这也说明, 为什么对阶是使小的阶码向大的阶码对齐。如果让大的阶码减小, 则尾数左移将丢失有效高位部分, 这显然是不允许的。

如果阶码以 2 为底, A_E 、 B_E 分别表示两数的阶码, A_M 、 B_M 分别表示两数的尾数, 则对阶操作如下:

- ① 如果 $A_E > B_E$, 则将操作数 B 的尾数右移一位, 阶码加 1, 记为: \bar{B}_M , $B_E + 1$, 直到 $B_E = A_E$ 。
- ② 如果 $A_E < B_E$, 则执行 \bar{A}_M , $A_E + 1$, 直到 $A_E = B_E$ 。

(3) 尾数相加/减

当两数的阶码对齐后, 相当于已将两数小数点实际位置对准, 然后让尾数相加/减, 结果送入 A_M , 记为: $A_M \pm B_M \rightarrow A_M$ 。

(4) 结果规格化

尾数加减后有可能不符合规格化的约定, 因而需要将尾数移位, 使之规格化, 并相应地调整阶码。这一操作称为结果规格化。为了便于判别是否符合规格化, 一种办法是让尾数的符号位扩展为双符号位, 分别定义为 A_{f1} 和 A_{f2} , 最高数位则定义为 A_1 。有两种需要规格化的情况, 分别讨论如下。

① 如果两同号数相加, 尾数有进位到符号位, 使 $|M| \geq 1$, 则需将尾数右移一位使之规格化, 称为右规。

逻辑条件: $A_{f1} \oplus A_{f2} = 1$

操作: \bar{A}_M , $A_E + 1$

② 如果两异号数相加 (或同号数相减), 使 $|M| < 1/2$, 则需将尾数左移一位使之规格化, 称为左规。

逻辑条件: $\bar{A}_{f1} \bar{A}_{f2} \bar{A}_1 + A_{f1} A_{f2} A_1 = 1$

操作: \bar{A}_M , $A_E - 1$

需要注意, 如果正尾数的最高有效位 A_1 不为 1, 应左移, 如 00.0101 左移为 00.1010; 如果补码表示的负尾数, 其 A_1 不为 0, 也应左移, 如 11.1011 左移为 11.0110。此外, 如果结果的尾数为 0, 则不管阶码为何值, 浮点数均为 0, 称为数量级零。显然, 这种情况不需要规格化。

由于浮点数的表示范围宽广, 在实际应用中很少出现溢出。理论上, 仅在两种极端情况下可能溢出。一种情况是同号数相加前, 其中一数的绝对值很大, 使正阶码已达到最大值, 而相加后又需右规, 则尾数右规时阶码增加, 有可能上溢。另一种情况是异号数相加前, 两数的绝对值很

小，使负阶码绝对值很大，而相加后又需左规，则尾数左规时阶码减少，有可能发生下溢。

2. 浮点乘法运算

设浮点数 $A = 2^{A_E} \cdot A_M$, $B = 2^{B_E} \cdot B_M$ ，则

$$A \times B = 2^{A_E + B_E} \cdot (A_M \times B_M) \quad (2-26)$$

即阶码相加，尾数相乘。其运算可按以下步骤进行。

① 检测能否简化操作，并置结果数符。如果操作数中有一个为 0，乘积必为 0，不需做其他操作。如果两数均不为 0，才进行乘法运算。结果数符按同号相乘为正、异号相乘为负的规则确定。

② 阶码相加。如果阶码用补码表示，则阶码相加可按常规补码加法进行。如果阶码用移码表示，由于阶码本身已有一个偏移量 2^m ，相加后偏移量将加倍，所以移码相加后应修正（即减去 2^m ）。阶码相加有可能产生溢出，同号相加可能上溢（正阶码），也可能下溢（负阶码）。当产生溢出时，浮点运算器将发出溢出信号，使机器转入溢出处理。

③ 尾数相乘，可以用任何一种定点小数乘法实现。可见浮点乘法包含两组定点运算，即定点整数的阶码运算与定点小数的尾数运算。这两组运算可以共用一个加法器，分步执行；也可以在常规加法器的基础上再设置一个专门的阶码加法器，并行执行，以提高运算速度。

④ 乘积规格化。尾数相乘后，可能需要左规。由于尾数是定点小数，相乘后不会出现需右规的情况。左规时阶码减 1，有下溢的可能。

3. 浮点除法运算

A 、 B 两数相除，其运算式如下：

$$A \div B = 2^{A_E - B_E} \cdot (A_M \div B_M) \quad (2-27)$$

即阶码相减，尾数相除。其运算可按以下步骤进行。

① 检测能否简化操作，并置商的数符。如果被除数为 0，则商为 0。如果除数为 0，不应相除，可给出标志信息表示除数为 0，另作处理。结果数符的置位规则与乘法相同。

② 尾数调整。检测被除数尾数的绝对值是否小于除数尾数的绝对值，以确保商的尾数为小数。如果不是，则将被除数尾数右移一位，并相应调整其阶码（阶码加 1）。在阶码调整时， A_E 有上溢的可能。

③ 阶码相减。如果阶码用补码表示，则减后不需修正。如果阶码用移码表示，则减后应加上 2^m 进行修正。因为移码相减后将丢失偏移量，所以应当通过修正来恢复偏移量。异号阶码相减时，有可能产生溢出。

④ 尾数相除。可用定点小数除法实现，如果有专门的阶码运算部件与尾数运算部件，则③和④两步可同时进行，否则需分步执行。

⑤ 结果不再规格化。由于两操作数均已规格化，即 $|M| \geq 1/2$ ，相除后其商（结果尾数）的绝对值必然大于等于 1/2，不需要左规。在进行尾数调整后，商（结果尾数）的绝对值必然小于 1，不需要右规。所以按上述操作产生的商不需要进行规格化处理。

2.4 常用的数据校验方法

通俗地讲，校验的方法是让写入的信息符合某种约定的规律，在读出时检验读出信息是否仍符合这一约定规律，如果符合，则基本上可判定读出信息正确无误。

现在使用的校验方法，大多采用冗余校验思想。因为待写入的二进制代码，从全 0 到全 1，

各种组合都有可能，不一定都符合规律。但是，可在写入时增加部分代码，称为校验位，将待写的有效代码和增加的校验位一起，按约定的校验规律进行编码，获得的编码称为校验码，全部写入主存。读出时，对读得的校验码（其中包括有效代码和增加的校验位）进行校验，看它是否仍满足约定的校验规律。对计算机工作本身所需的有效代码而言，校验位是为校验需要而额外增加的，称为冗余位。如果校验规律选择得当，不仅能查明是否有错，而且可根据出错特征判定最大可能是哪一位出错，从而将其变反纠正，称为纠错。

为了判断一种校验码码制的冗余程度，并估价它的查错能力与纠错能力，提出了“码距”这样一个概念。

由若干位代码组成一个字，称为码字。一种编码体制（码制）中可有多种码字。将两个不同的码字逐位比较，代码不同位的个数称为这两个码字间的“距离”。在一种码制中，任何两个码字间的距离可能不同，各合法码字（非出错的码字）间的最小距离称为这种码制的“码距”。例如，常用的 8421 码是一种编码体制，0000 与 0001 之间距离为 1，而 0000 与 1111 之间距离为 4。因此 8421 码的码距为 1。如果从主存中读得一个码字为 0111，就无法判断它是正确的 7，还是 6 的最低位出错。因此，认为 8421 码的码距为 1 太小，只能区分两个合法码字的不同，不具备查错能力，更不具备纠错能力。

如果按照某一校验规律编码，可使其码距扩大。因为增加校验位（冗余位）之后，代码组合数增加，但我们只取其中符合校验规律的合法代码，将不符合校验规律的视为出错代码。因此从信息量角度看，合法代码之间的距离加大，就有可能分辨合法代码与出错代码，并有可能判断该出错代码靠近哪个合法代码，因而确定可能是哪位出错，将它变反纠正为正确的合法代码。

综上所述，约定的校验规律提供了编码与校验（译码）的基本依据，而扩大码距则从扩大信息量的角度提供了查错与纠错的可能性。

2.4.1 奇偶校验

主存储器一般都支持奇偶校验，这是一种简单且应用广泛的校验方法。

（1）奇偶校验原理

奇偶校验是根据代码字的奇偶性质进行编码和校验，有两种校验规则：① 奇校验，使完整编码（有效位和校验位）中“1”的个数为奇数个；② 偶校验，使完整编码（有效位和校验位）中“1”的个数为偶数个。

有效信息本身不一定满足约定的奇偶性质，但增设校验位后可使编码符合约定的奇偶性质。如果两个有效信息代码字之间有一位不同（至少有一位不同），则它们的校验位也应不同，因此奇偶校验码的码距为 2。从码距看，奇偶校验能发现一位错，但不能判断是哪位出错，所以没有纠错能力。从所采用的奇偶校验规则看，只要是奇数个代码出错，都将破坏约定规律，所以这种校验方法的查错能力为：能发现奇数个错。如果是偶数个错，不影响码字的奇偶性质，因此不能判断信息是否出错。

| | |
|-----------------|-----------|
| 【例 2-56】 待编有效信息 | 10110001 |
| 奇校验码（配校验位后） | 101100011 |
| 偶校验码（配校验位后） | 101100010 |

| | |
|-----------------|-----------|
| 【例 2-57】 待编有效信息 | 10110101 |
| 奇校验码（配校验位后） | 101101010 |
| 偶校验码（配校验位后） | 101101011 |

(2) 奇偶校验逻辑

为了快速地进行奇（偶）编码写入与读出后的奇（偶）校验，多采用并行奇偶统计方法，其逻辑电路可用若干异或门构成，如图 2-10 所示。这种塔形结构同时给出了“奇形成”、“奇校验”、“偶形成”、“偶校验”。如果机器选用偶校验方式，可取消“奇形成”和“奇校验”两个信号。

现以偶校验为例说明它的编码、译码过程。

① 编码。编码即写入时配置校验位，当将 8 位代码 $D_7 \sim D_0$ 写入主存时，同时将它们送往偶校验逻辑电路。

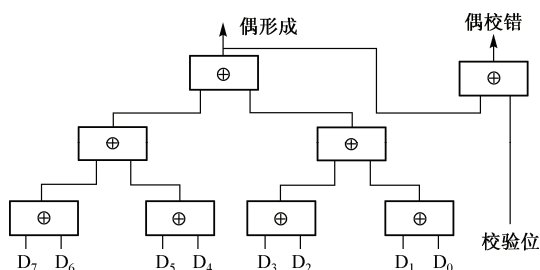


图 2-10 奇偶校验逻辑电路

若 $D_7 \sim D_0$ 中有偶数个 1，则 $D_7 \oplus D_6 \oplus D_5 \oplus D_4 \oplus D_3 \oplus D_2 \oplus D_1 \oplus D_0 = 0$ ，即“偶形成”=0。

若 $D_7 \sim D_0$ 中有奇数个 1，则 $D_7 \oplus D_6 \oplus D_5 \oplus D_4 \oplus D_3 \oplus D_2 \oplus D_1 \oplus D_0 = 1$ ，即“偶形成”=1。

将 $D_7 \sim D_0$ 和“偶形成”一道写入主存。

② 译码。译码即读出时进行校验，将读出的 8 位代码与 1 位校验位同时送入偶校验逻辑电路，若“偶校验”为 0，表明数据正确（无奇数个错）；若“偶校验”为 1，表明数据有错（奇数个错）。因此，“偶校验”就是检错信息。

奇偶校验是一种编码校验，在主存储器中是按字节（字）为单位进行的，基本上依靠硬件实现。除此之外，还可通过软件实现“累加和”校验，即在写入一个数据块或程序段时，边写入边累加，最后将累加和也写入主存。如果需要写后复查，或是在调用该程序段（或数据块）前先检查信息是否被破坏，可以通过检查累加和实现校验。在执行程序的过程中，信息可能被修改，累加和关系也将被更改。因此，累加和校验只能应用于有限场合。

2.4.2 海明校验

这是由 Richard Hamming 提出的一种校验方法，因而称为海明校验。它实际上是一种多重偶校验，即将代码按照一定规律组织为若干小组，分组进行奇偶校验，各组的检错信息组成一个指误字，不仅能检测是否出错，而且在只有一位出错的情况下可指出是哪一位错，从而将该位自动地变反纠错。下面通过一个例子说明其编码方法、查错与纠错能力。

【例 2-58】 待编信息 4 位 $A_1A_2A_3A_4$ 进行海明校验编码后，要求能发现并纠正一位错。

(1) 分成几组？增设多少校验位？

设待编的有效信息 k 位，分成 r 组，每组增设一个校验位，共需增设 r 位校验位，组成一个 n 位的海明校验码。校验时每组产生一位校验信息，组成一个 r 位的指误字，可指出 2^r 种状态，其中全 0 表示无错，余下的组合可分别指明 $2^r - 1$ 位中的某一位错误。因此，从信息量的角度，如果要求海明校验码能发现并纠正一位错，则应满足下述关系：

$$n = k + r \leq 2^r - 1 \quad (2-28)$$

若 $k=4$ ，则 $r \geq 3$ ，组成 7 位海明码。

(2) 分组方法

如本例，待编有效信息 $A_1A_2A_3A_4$ ，增设校验位 $P_1P_2P_3$ ，分为 3 组校验，可产生 3 位指误字 $G_3G_2G_1$ 。为了使指误字指明出错的位，就要求二者之间存在唯一的对应关系。本例采取一种最简单的对应关系；让指误字代码与出错的位序号相同。例如，将 A_1 安排在第 3 位，让它参加第 1 组与第

2 组的校验。将来如果是第 3 位出错，则 A_1 未参加的第 3 组检错信息为 0，而 A_1 参加的第 1、2 组检错信息均为 1，于是产生指误字 $G_3G_2G_1=011$ ，说明是第 3 位出错。按照这一分组原则，7 位海明校验码的序号与分组关系如表 2-9 所示，以符号“√”标注每位参加哪些组。

表 2-9 7 位海明编码 ($k=4, r=3, d=3$)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 指误字 |
|-------|-------|-------|-------|-------|-------|-------|-------|-----------------|
| | P_1 | P_2 | A_1 | P_3 | A_2 | A_3 | A_4 | |
| 第 3 组 | | | | √ | √ | √ | √ | G_3 |
| 第 2 组 | | √ | √ | | | √ | √ | G_2 |
| 第 1 组 | √ | | √ | | √ | | √ | G_1 |
| 正确码 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | $G_3G_2G_1=000$ |
| 一位错 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | $G_3G_2G_1=101$ |

每个校验位只参加一组奇偶校验，将 $P_1P_2P_3$ 分别安排在第 1、2、4 位，并分别参与相应的第 1、2、3 组的校验。这是因为每个校验位只是为某一组奇偶校验而设置的，与其他组无关，所以只参与一组校验。换句话说，每组只有一位校验位。由于要求指误字能反映出错位的序号，所以将 P_1 安排在第 1 位。只参加第 1 组校验，一旦 P_1 出错，错误字为 001 表明第 1 位出错。同理， P_2 安排为第 2 位，只参加第 2 组； P_3 安排为第 4 位，只参加第 3 组。

有效信息 $A_1A_2A_3A_4$ 分别参加两组以上的校验，它们依次占据剩下的 3、5、6、7 位，并分别参加与此相应的组别：(1, 2)，(1, 3)，(2, 3)，(1, 2, 3)。由于所有各位参加的组别并不完全重复，一旦某位出错，指误字将与出错位序号存在唯一的对应关系。

(3) 编码

假定每组都采取偶校验，即让 1 的个数为偶数。现欲向外存写入有效信息 (1010)，将它们分别填入第 3、5、6、7 位，即 $A_1A_2A_3A_4$ ，再分组进行奇偶统计，按偶校验要求填入校验位： $P_1P_2P_3$ 。如果参加第 1 组偶校验的有： $P_1A_1A_2A_4$ ， P_1 应为 1，才能使该组 1 的个数为偶数。同理， $P_2=0$ ， $P_3=1$ 。最后得到海明校验码 7 位 (1011010) 见表 2-9。

(4) 查错与纠错 (译码)

如果从外存中读得 7 位海明码为 1011010，按表 2-9 分组进行奇偶检测。三组都满足偶校验要求， $G_3G_2G_1=000$ ，表明收到的码字是正确的，可从中提出有效信息 $A_1A_2A_3A_4$ 。

如果从外存中读得的代码是 1011110，同样按表 2-9 分组进行奇偶检测，得到三组的检错信息，形成指误字 $G_3G_2G_1=101$ ，表明第 5 位 (即 A_2) 出错。将第 5 位变反，即可纠正为 1011010。

由于海明校验的实质是分组奇偶校验，因此它的编码和查错逻辑与图 2-10 相似。但海明校验具有自动纠错能力，可将三位检错信息构成的指误字 $G_3G_2G_1$ 进行译码，除全 0 外，其余 7 种译码输出分别控制七路异或门，控制对应位读出信息是否需要变反纠正。如例 2-58，第 5 位异或门的一路输入为 A_2 ，另一路输入由指误字译码器提供 1， $1 \oplus A_2 = \bar{A}_2$ ，变反纠正后输出。限于篇幅，不再画出逻辑电路，读者可按上面所述自行设计。

在 $k=4, r=3$ 的海明码中，两合法 (正确) 码字之间至少有一位有效信息不同，由于有效信息位 A_i 至少参加两组校验，相应的两组校验位也将随之不同，因此这种海明码的码距 $d=3$ 。进一步分析，可以得出结论： $d=3$ 的海明校验码，可检测出 $2 \times (d-1)$ 位错，或用来检测并纠正 1 位错。在设计系统时应事先确定；要么只要求发现错误而不纠正，则可检测出 1 位出错和 2 位出错 (大家可以自举一个 2 位错实例)；要么要求发现并自动纠正 1 位错，如例 2-58。

能否做到检测 2 位错并纠正 1 位错呢？为了提高校验码的查错、纠错能力，就需要进一步扩

大码距。一种方案是：增加一个第 4 组，所有各位 ($P_i A_i$) 都参加这组校验，则该组的检错信息 G_4 将能判别是 1 位错 ($G_4=1$)，还是 2 位错 ($G_4=0$)。如果是 1 位错，则由指误字译码输出将其变反纠正。如果是 2 位错，即 $G_4=0$ ，而 $G_3 G_2 G_1 \neq 0$ 则只给出校验错信息，不予纠正。

在 $k=4$ 、 $r=3$ 海明码中，待编有效信息 4 位，而校验位 3 位，冗余度较大，将降低外存的有效利用率。如果增大 k ，则 r 增加不多，冗余度降低。但 k 增大导致硬件增加。一般让 $k=8$ ，即以字节为单位进行海明编码。

海明校验至今仍是一种基本的校验方法，用于要求快速自动纠错的场合。

2.4.3 循环冗余校验

循环冗余码校验 (Cyclic Redundancy Check, CRC) 是目前在磁表面存储器中应用最广泛的一种校验方法，也是多机网络通信中常用的校验方法。它所约定的校验规则是：让校验码能为某一约定代码所除尽。如果除得尽，表明代码正确；如果除不尽，余数将指明出错位所在位置。

任意一串数码，很可能除不尽，将产生一个余数。如果让被除数减去余数，势必能为约定除数所除尽。但减法操作可能需要借位运算，难以用简单的拼装方法实现编码。因此我们采用一种模 2 运算，即通过模 2 减实现模 2 除，以模 2 加将所得余数拼接在被除数后面，形成一个能除尽的校验码。当然，在采用模 2 除后，对除数的选择是有条件的。

这里所讲的模 2 运算是一种以按位加减为基础的四则运算，不考虑进位和借位。注意，它与以 2 为模的定点小数运算是两个不同的概念。因此，模 2 加减即按位加减，也就是异或，可用异或门实现。

待编码的信息是一串代码，可能是表示数值大小的数字，也可能是字符编码，或其他性质的代码。在模 2 除中，暂将它视为数字，可用多项式来描述。我们定义待编信息（被除数）为 $M(x)$ ；约定的除数为 $G(x)$ ，因为它用来产生余数的，所以 $G(x)$ 又称为生成多项式，所产生的余数 $R(x)$ 相当于所配的冗余校验位。

(1) 编码方法

① 将待编码的 k 位有效信息 $M(x)$ 左移 r 位，得 $M(x) \cdot x^r$ 。这样做的目的是空出 r 位，以便拼装将来求得的 r 位余数。

② 选取一个 $r+1$ 位的生成多项式 $G(x)$ 。对 $M(x) \cdot x^r$ 作模 2 除。

$$\frac{M(x)x^r}{G(x)} = Q(x) + \frac{R(x)}{G(x)} \quad (\text{模 2 除})$$

要产生 r 位余数，所以除数应为 $r+1$ 位。

③ 将左移 r 位的待编有效信息，与余数 $R(x)$ 作模 2 加（减），即拼接为循环校验码。

$$M(x) \cdot x^r + R(x) = Q(x) \cdot G(x) \quad (\text{模 2 加})$$

在按位运算中，模 2 加与模 2 减的结果是一致的，所以 $M(x) \cdot x^r - R(x) = M(x) \cdot x^r + R(x)$ 。 $M(x) \cdot x^r$ 的末尾 r 位是 0，所以与余数 $R(x)$ 的加减实际上就是将 $M(x)$ 与 $R(x)$ 相拼接。拼接成的校验码必定能为约定的 $G(x)$ 所除尽。在本节中，将 $M(x) \cdot x^r + R(x)$ 称为循环校验码。但在许多磁表面存储器的记录格式中，常只将 $R(x)$ 部分称为校验码。

【例 2-59】 将 4 位有效信息 1100 编成循环校验码，选择生成多项式 1011。

$$\begin{array}{lll} M(x) = x^3 + x^2, & \text{即 } 1100 & (k=4) \\ M(x) \cdot x^r = x^6 + x^5 & \text{即 } 1100000 & (r=3) \\ G(x) = x^3 + x + 1 & \text{即 } 1011 & (r+1=4) \end{array}$$

$$\frac{M(x) \cdot x^3}{G(x)} = \frac{1100000}{1011} = 1110 + \frac{010}{1011} \quad (\text{模 } 2 \text{ 除})$$

$$M(x) \cdot x^3 + R(x) = 1100000 + 010 = 1100010 \quad (\text{模 } 2 \text{ 加})$$

我们将编好的循环校验码称为(7, 4)码, 即 $n = 7$, $k = 4$ 。

(2) 译码与纠错

将收到的循环校验码用约定的生成多项式 $G(x)$ 去除, 如果码字无误, 则余数为 0; 如果某一位出错, 则余数不为 0。不同位出错则余数不同, 余数代码与出错位序号之间有唯一的对应关系。

通过上例可求出其出错模式如表 2-10 所示, 即余数与出错位序号之间的对应模式。更换不同待测码字可以证明, 出错模式只与码制和生成多项式有关, 与码字代码无关, 对于(7, 4)CRC 码, 表 2-10 具有通用性, 可作为(7, 4)码的判别依据。当然, 对于其他码制或选用其他生成多项式, 出错模式可能不同。

表 2-10 中列举了 8 种情况。一种是正确码字, 除后余数为 0。其余 7 种是依次有 1 位出错, 余数不为 0, 与出错位序号有唯一的对应模式。深入研究将发现一个有实用价值的规律, 如果有一位出错, 用 $G(x)$ 除后得到一个不为 0 的余数, 若对该余数补 0, 继续除, 各次余数将按表 2-10 顺序循环。例如, 第 7 位 A_7 出错, 余数 001, 补 0 后继续除, 得余数 010, 以后将依次为 100, 011, 110, 111, 101, 然后是 001, 呈循环状。这就是“循环码”名称的由来。

表 2-10 (7, 4)循环码的出错模式 ($G(x) = 1011$)

| | A_1 | A_2 | A_3 | A_4 | A_5 | A_6 | A_7 | 余 数 | 出错位 |
|----|-------|-------|-------|-------|-------|-------|-------|-------|-----|
| 正确 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 0 0 | 无 |
| 出错 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 0 1 | 7 |
| | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 1 0 | 6 |
| | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 0 0 | 5 |
| | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 1 1 | 4 |
| | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 1 0 | 3 |
| | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 1 1 | 2 |
| | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 0 1 | 1 |

这一规律启示我们, 可以采取一种节省硬件的纠错办法: 只设置余数 101 的译码输出, 对应于第 1 位出错位置; 如果校验后发现余数不为 0, 一边对余数补 0 继续作模 2 除, 同时将被检测码字循环左移一位; 当出现余数 101 时, 出错位也移至第 1 位位置, 可用异或门将之变反纠正, 或在由第 1 位移往第 7 位的途中予以纠正; 继续移满一个循环, 即共移 7 次将得到一个纠正后的正确码字。这样就不必像海明校验那样为每一位提供纠正条件。当位数增多时, 采用循环码能有效地降低硬件成本。循环码校验可用硬件逻辑实现, 如在磁表面存储器中; 也可以用软件实现, 如在某些通信网中。

(3) 生成多项式的选取

并不是任何一个多项式都可以作为生成多项式的, 从检错和纠错的要求出发, 生成多项式应能满足下列要求: 任何一位发生错误都应使余数不为 0, 不同位发生错误应当使余数不同, 应满足余数循环规律。

对使用者来说, 可从有关资料上查到对应于不同码制的可选生成多项式。计算机和通信系统中广泛使用两种标准: CCITT (国际电报电话咨询委员会) 推荐 $G(x) = x^{16} + x^{15} + x^2 + 1$; IEEE (美国电气和电子工程师学会) 推荐 $G(x) = x^{16} + x^{12} + x^5 + 1$ 。

循环校验的硬件逻辑的核心是模 2 除线路，限于篇幅，不再详述。下面仅给出一个模 2 除的算式，供参考。

$$\begin{array}{r}
 \begin{array}{r}
 1110 \\
 1011 \overline{) 1100000} \\
 \underline{1011} \\
 1110 \\
 \underline{1011} \\
 1010 \\
 \underline{1011} \\
 0010 \\
 \underline{0000} \\
 010
 \end{array}
 \end{array}
 \begin{array}{l}
 \hline Q(x) \\
 \hline M(x) \cdot x^3 \\
 \hline \\
 \hline R(x)
 \end{array}$$

当被除数最高位为 1 时，商 1，与 $G(x)$ 作模 2 减，所得余数左移一位。当被除数最高位为 0 时，商 0，余数左移一位。

习 题 2

1. 简要解释下列名词术语

位权 基数 真值 机器数 原码 补码 定点数 浮点数
规格化浮点数 ASCII 码 算术移位 逻辑移位

2. 将二进制数 $(1111010.00111101)_2$ 转换为八进制数和十六进制数。

3. 将二进制数 $(101010.01)_2$ 转换为十进制数与 BCD 码。

4. 将八进制数 $(37.2)_8$ 转换为十进制数与 BCD 码。

5. 将十六进制数 $(AC.E)_{16}$ 转换为十进制数与 BCD 码。

6. 将十进制数 $(75.34)_{10}$ 转换为 8 位二进制数、八进制数及十六进制数。

7. 将十进制数 $13/128$ 转换为二进制数。

8. 分别写出下列各二进制数的原码与补码，字长（含一位数符）为 8 位。

(1) 0 (2) ~ 0 (3) 0.1010

(4) ~ 0.1010 (5) 1010 (6) ~ 1010

9. 若 $X_{\text{补}} = 0.1010$ ，写出其 $X_{\text{原}}$ 与真值 X 。

10. 若 $X_{\text{补}} = 1.1010$ ，写出其 $X_{\text{原}}$ 与真值 X 。

11. 某定点小数字长 16 位，含 1 位符号，原码表示，分别写出下列典型值的二进制代码和十进制真值。

(1) 非零最小正数 (2) 最大正数
(3) 绝对值最小负数 (4) 绝对值最大负数

12. 某定点小数字长 16 位，含 1 位符号，补码表示，分别写出下列典型值的二进制代码和十进制真值。

(1) 非零最小正数 (2) 最大正数
(3) 绝对值最小负数 (4) 绝对值最大负数

13. 某定点整数字长 16 位，含 1 位符号，补码表示，分别写出下列典型值的二进制代码和十进制真值。

(1) 非零最小正数 (2) 最大正数
(3) 绝对值最小负数 (4) 绝对值最大负数

14. 某浮点数字长 16 位，其中阶码 6 位，含 1 位阶符，补码表示，以 2 为底；尾数 10 位，含 1 位数符，补码表示，规格化。分别写出下列各典型值的二进制代码和十进制真值。

(1) 非零最小正数 (2) 最大正数

(3) 绝对值最小负数

(4) 绝对值最大负数

15. 若采用图2-4的浮点数格式, 字长 16 位, 其中阶码 6 位, 含 1 位阶符, 补码表示, 以 2 为底; 尾数 10 位, 含 1 位数符, 补码表示, 规格化; 某浮点数代码为(A27F)₁₆, 写出其十进制真值。

16. 若采用图2-5所示的 IEEE754 短浮点数格式, 请将十进制数 37.25 写成浮点数, 并写出其二进制代码序列, 再转换成 16 进制数。

17. 用变形补码计算 $X_{补} + Y_{补} = ?$ 并指出是否有溢出。

(1) $X_{补} = 00.110011$ $Y_{补} = 00.101101$

(2) $X_{补} = 00.010110$ $Y_{补} = 00.100101$

(3) $X_{补} = 11.110011$ $Y_{补} = 11.101101$

(4) $X_{补} = 11.001101$ $Y_{补} = 11.010011$

18. 用变形补码计算 $X_{补} - Y_{补} = ?$ 并指出是否有溢出。

(1) $X_{补} = 00.100011$ $Y_{补} = 00.101101$

(2) $X_{补} = 00.110110$ $Y_{补} = 11.010011$

(3) $X_{补} = 11.100011$ $Y_{补} = 00.110100$

(4) $X_{补} = 11.101101$ $Y_{补} = 11.010011$

19. 用流程图描述下列算法流程。

(1) 补码一位乘法

(2) 原码两位乘法

(3) 原码加减交替除法

(4) 补码加减交替除法

(5) 浮点加减运算

(6) 浮点乘法运算

(7) 浮点除法运算

20. 参照图 2-9 形式, 设计下列乘法器和除法器。

(1) 补码一位乘的乘法器

(2) 原码两位乘的乘法器

(3) 原码加减交替除法器

(4) 补码加减交替除法器

21. 某乘法器基本字长 8 位 (含 1 位数符), 运算时可扩展为双符号位。请参照图 2-9 结构, 画出乘法器的完整逻辑图。

22. 欲写入代码 10011100 (原始有效信息):

(1) 采用奇校验, 写出配校验位后的校验码。

(2) 采用偶校验, 写出配校验位后的校验码。

23. 欲写入 8 位有效信息 01101101, 试将它编为海明校验码。以表格形式说明其编码方法, 并分析所选用的编码方案具有什么样的检错与纠错能力。

24. 某海明编码 $K=4$, $r=3$, 请为此设计其编码、译码、纠错逻辑。

25. 某循环校验码, 生成多项式 $X^8+X^4+X^0$ 。请为此设计一个模 2 除法器。

26. 设计一套硬件逻辑, 以实现循环校验码的编码、译码、校正。画出粗框图。

27. 设计一套子程序, 以实现循环校验码的编码、译码、校正。画出流程图。

28. 将 4 位有效信息 1001 编成循环校验码, 选择生成多项式 $X^3+X^1+X^0$, 试写出编码过程。

第3章 CPU 子系统

在现代计算机中，传统的运算器和控制器已逐渐合为一个整体，成为计算机系统的核心组成部分，称为 CPU（Central Processing Unit，中央处理器）。在微型计算机中，CPU 被集成在一块称为微处理器的芯片上，通过内部总线，建立起芯片内各部件之间的信息传送通路。而在大型、巨型计算机中，为支持大量的复杂计算，运算器目前仍需由多块芯片构成，与控制器保持相对独立。随着并行处理技术的发展，用多个微处理器所构成的多机系统来实现大型机、巨型机的功能，成为了一种主流的发展趋势。

本章的学习目的是建立起 CPU 这一层次的整机概念，体现在 CPU 的逻辑组成和工作机制这两方面。CPU 的主要功能是执行指令，控制各项操作，包括运算操作、传送操作、输入/输出操作等。为了实现这些功能，需要解决下面几个关键问题：

- ◎ CPU 由哪些部件组成？
- ◎ 各部件之间如何交换信息？
- ◎ CPU 如何建立与外部的连接？
- ◎ CPU 如何形成控制命令（微命令）序列，以控制指令的执行？

前三个问题涉及 CPU 的逻辑组成，最后一个问题涉及 CPU 的工作机制。为此，本章先介绍 CPU 的基本组成；再讨论指令系统以及运算部件的组织；并通过一个 CPU 模型，重点讨论 CPU 的数据通路结构和指令的执行过程，从寄存器传送级来分析指令的分步执行流程，从微操作控制级来阐明寄存器传送的具体实现。

3.1 概述

3.1.1 CPU 的基本组成

CPU 的种类繁多，体系架构也各有不同，但通常都包含了运算部件、寄存器组、微命令产生部件、时序系统等基本部件，这些部件通过 CPU 内部总线连接，实现信息交换。其中，运算部件和一部分寄存器属于传统运算器部分，另一部分寄存器、微命令产生部件、时序系统等则属于传统控制器部分。

1. 运算部件

运算部件的基本任务是对操作数进行加工处理。那么，如何获取操作数呢？怎样对数据进行运算操作的？又如何输出运算结果的呢？在回答这些问题之前，先分析运算部件的基本组成情况和结构逻辑，如图 3-1 所示。

（1）输入逻辑

操作数既可以来自各种寄存器，也可以来自 CPU 内部的数据线。每次运算最多只能对两个数据进行操作，所以运算部件设置了两个输入逻辑，它们可以是选择器或暂存器，两者的主要作用是分别选择两个操作数以便进行后续运算。

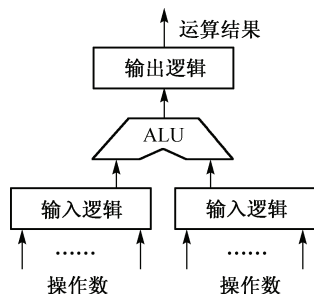


图 3-1 基本运算部件组成

(2) 算术逻辑运算部件 (ALU)

ALU (Arithmetic Logic Unit) 是运算部件的核心, 完成具体的运算操作。它的主要部件就是一个加法器, 负责对两个操作数进行求和运算。两个数进行算术加时有可能产生进位, 所以加法器除了具有求和逻辑外, 还具有提供进位信号的传递逻辑, 称为进位链。

(3) 输出逻辑

运算结果可以直接送往接收部件, 也可以经左移或右移后再送往接收部件, 所以输出逻辑往往具有移位功能, 常采用移位门 (多路选择器), 通过斜位传送实现左移、右移或者字节交换等操作。运算部件直接影响计算机的运算功能, 有 4 种典型的 ALU 设置方式。

① 只设置一个 ALU。在低档的微处理器中, 只设置一个 ALU, 因而通过硬件只能实现基本的定点加、减运算和逻辑运算, 依靠软件子程序可实现定点乘除运算、浮点运算和其他更复杂的运算。

② 设置一个 ALU, 并配合时序控制。在高档微处理器中, 除设置一个 ALU 外, 还可通过时序控制, 在硬件级实现定点乘法和除法运算, 通常分为若干步来完成。如果设置了专门的阵列乘法器和除法器, 则乘、除运算也可同加、减运算一样, 只需一步便能完成。

浮点运算较为复杂, 可采用软件或硬件方法来实现, 如执行浮点运算子程序; 或者选择配置浮点协处理器, 将它作为一种扩展部件, 通过简单的接口芯片连接到 CPU 上, 或做成一种浮点加速器插件, 直接插入到系统总线的插槽中。

③ 设置一个 ALU, 并将定点乘除部件和浮点部件作为基本配置。在超级小型机中, 这是一种常见的配置情况, 机器的运算功能已达到传统的中型机范畴。

④ 设置多个运算部件。大型、巨型计算机中往往有多个运算部件, 以实现流水处理, 完成复杂的运算操作。例如, 巨型机 CRAY-1 有 12 个运算部件, 包括定点标量运算器、浮点运算器、向量运算器等, 分别用来处理标量运算、向量运算。

2. 寄存器组

计算机在工作时, CPU 需要处理大量的控制信息和数据信息, 如: 对指令进行译码以产生相应的控制信号; 对操作数进行算术或逻辑运算并根据运算结果决定后续操作等。因此在 CPU 内部, 需要设置若干寄存器来暂时存放这些信息。根据承担的功能任务不同及存储信息类型的差异, 这些存储器总体上可分为如下几种。

(1) 通用寄存器组

这是一组可通过程序访问的寄存器, 在指令系统中还为这组寄存器分配了各自的编号, 因而可以编程访问某个指定编号的寄存器。

这些寄存器自身的逻辑一般都很简单, 也比较统一, 甚至可以是一些小规模存储单元。但通过编程以及与运算部件的相互配合, 可以实现多种功能, 如提供操作数和存放临时行的运算结果, 或者作为地址指针, 或者作为基址寄存器、变址寄存器、计数器等。因为这类寄存器通用性较强, 故常把它们称为通用寄存器。

不同的计算机对通用寄存器的功能分配并不完全相同。有的计算机没有对各通用寄存器进行特定的任务分工, 它们可被指定去完成各种工作。相应地, 通用寄存器组的命名也就没有特殊意义, 如可简单地命名为 R0、R1、R2 等。有的计算机则为各通用寄存器分别规定了特定的任务, 并按任务对其命名, 如累加器 AX、基址寄存器 BX、计数寄存器 CX 和数据寄存器 DX 等。

从寄存器的组成来看, 早期用 D 触发器构成寄存器组, 各寄存器可以同时输入或输出数据,

但采用的是小规模集成电路，集成度很低；现在广泛采用中规模集成的 RAM 构成寄存器组，一个存储单元用作一个寄存器。单口（单地址端、单数据端）RAM 每次只能访问一个寄存器；双口（双地址端、双数据端）RAM 则每次可以访问两个寄存器，满足了双操作数运算的需要。

（2）暂存器

与通用寄存器不同，暂存器没有编号，不能被编程访问，只能在 CPU 工作时内部专用。设置暂存器的目的是暂存某些中间过程产生的信息，以免覆盖了通用寄存器的内容。例如，要将某个主存单元的内容送往另一个主存单元，需先从主存读取源数据，并将该内容暂存起来，再根据目的单元地址，将该内容写入目的主存单元。如果将读出的源数据暂存在某个通用寄存器中，可能破坏该寄存器原来的内容，因此需将源数据暂时存放在暂存器中，这个中间暂存过程对 CPU 外部是透明的。又如，当两个操作数不能同时送往 ALU 时，也需要在 ALU 的输入端设置一个暂存器来先暂存一个操作数，当两个操作数都准备好，才同时送入 ALU 进行运算。

以上两种寄存器主要用来存放数据信息，提供处理对象。

（3）指令寄存器

指令寄存器（Instruction Register, IR）用来存放当前正在执行的指令，它的输出包括操作码信息、地址信息等，是产生微命令的主要逻辑依据。

为了提高读取指令的速度，常常在主存的数据寄存器与指令寄存器之间建立直接传输通路，指令从主存取出后经数据寄存器，沿直接通路快速送往 IR。为了提高指令之间的衔接速度、支持流水线操作，大多数计算机将指令寄存器扩充为指令队列（也叫指令栈），允许预取若干条指令。

（4）程序计数器

程序计数器（Program Counter, PC），也称为指令计数器或指令指针，用来指示指令在存储器中的存放位置。当程序顺序执行时，每次从主存取出一条指令，PC 内容就增量计数，指向下一条指令的地址。增量值取决于现行指令所占的存储单元数，如果现行指令只占 1 个存储单元，则 PC 内容加 1；若现行指令占了 2 个存储单元，那么 PC 内容加 2。当程序要转移时，将转移地址送入 PC，使 PC 指向新的指令地址。因此，当现行指令执行完时，PC 中存放的总是后续指令的地址，将该地址送往主存的地址寄存器，便可从主存储器读取到下一条指令。

（5）程序状态字寄存器

程序状态字（Program Status Words, PSW）寄存器，用来记录现行程序的运行状态和指示程序的工作方式。PSW 的内容主要有以下两部分。

① 特征位，也称为标志位、条件码，用来反映当前程序的执行状态。一条指令执行后，CPU 根据执行结果设置相应特征位，作为决定程序流向的判断依据。例如，如果特征位的状态与转移条件符合，程序就进行转移；如果不符合，则顺序执行。常见的特征位包括：

- ⊙ 进位位 C——运算后如果产生进位，将 C 置为 1，否则将 C 清为 0。
- ⊙ 溢出位 V——运算后如果产生溢出，将 V 置为 1，否则将 V 清为 0。
- ⊙ 零位 Z——运算结果为零，将 Z 置为 1，否则将 Z 清为 0。
- ⊙ 负位 N——运算结果为负数，将 N 置为 1，否则将 N 清为 0。
- ⊙ 奇偶位 P——代码中 1 的个数为奇数，将 P 置为 1，否则将 P 清为 0。

② 编程设定位。PSW 中的某些位或某些字段是由 CPU 编程设定的，以决定程序的调试、对中断的响应、程序工作方式等，包括：

- ⊙ 跟踪位 T——编程设定的断点标志，以便对程序进行调试。如果在编程时将 T 置为 1，并在程序断点处安排一条测试指令，则执行到该条指令时，检测 T 的状态，便可由当前程

序转入调试程序。

- ◎ 允许中断位 I 或程序优先级字段——在程序运行过程中，当出现外部中断请求时，CPU 是否暂停当前程序去响应中断请求呢？这需要比较现行程序和外部请求事件的重要性。因此，有的计算机在 PSW 中设置了允许中断位 I。当 I 为 1 时，表明外部请求事件比现行程序重要，允许 CPU 响应外部中断请求；当 I 为 0 时，则表明现行程序更重要，CPU 不能响应外部请求。有的计算机在 PSW 中设置优先级字段，编程时为现行程序赋予某种优先级别，外部请求也分成不同的优先级。如果外部请求的优先级高于现行程序的优先级，CPU 响应外部请求，否则不响应。
- ◎ 工作方式字段——工作方式可用来指明程序的特权级。例如，有些计算机将 CPU 状态分为用户态与核心态（又称为管态），在用户态下，CPU 运行用户程序，程序中不能使用某些特权指令；在核心态下，CPU 运行操作系统等系统管理软件，允许使用某些特权指令。

不同的计算机在 PSW 的设置上可能差别较大。例如，有些微处理器中只设置若干位特征标志，并没有形成 PSW 的整体概念。某些计算机的 PSW 中还有其他更复杂的信息。

IR、PC、PSW 等寄存器属于控制部件，用来存放控制信息。

（6）地址寄存器

CPU 访问主存时，首先要找到需访问的存储单元，因此设置地址寄存器（Memory Address Register, MAR）来存放被访单元的地址。当需要读取指令时，CPU 先将 PC 的内容送入 MAR，再由 MAR 将指令地址送往主存进行寻址。当需要读取或存放数据时，也要先将该数据的有效地址送入 MAR，再送往主存进行译码。

（7）数据缓冲寄存器

数据缓冲寄存器（Memory Buffer Register, MBR）用来存放 CPU 与主存之间交换的数据。由 CPU 写入主存的数据通常先送入 MBR，然后从 MBR 送往主存相应单元。由主存读出的数据一般也先送入 MBR，再从 MBR 送至 CPU 指定的寄存器。

MAR 和 MBR 是连接 CPU 与主存的桥梁，设置这两个寄存器使 CPU 与主存之间的传输通路变得比较单一，容易控制。这两个寄存器不能直接编程访问，对用户是透明的。例如，一条传送指令将 CPU 某寄存器的内容送入某存储单元，用户所看到的操作仅仅是该寄存器和该存储单元之间的数据传输。这一数据传输的具体实现过程，即通过 MAR 的地址发送和经过 MBR 的数据中转等细节，用户则是看不见的。

3. 控制器及控制方式

控制部件的功能主要是负责对指令进行译码，并且发出为完成每条指令所要执行的操作的控制信号。

从用户的角度来看，计算机的工作体现为指令序列的连续执行；从内部实现机制来看，指令的读取和执行又体现为信息的传输，相应地，在计算机中形成了控制流和数据流这两大信息流。实现信息传送要靠微命令的控制，因此在 CPU 中设置微命令产生部件，根据控制信息产生微命令序列，对指令功能所要求的数据传送进行控制，并在数据传送至运算部件时控制完成运算处理。

可执行程序的最终形态是指令序列，各条指令往往需分步执行，如一条运算指令的读取和执行过程常需划分为取指令、取源操作数、取目的操作数、执行运算操作、存放运算结果等阶段，每个阶段又可再分成若干步操作，这就要求微命令也能分步产生。因此，微命令产生部件在一段时间内发出一组微命令，控制完成一步操作；在下一段时间内又发出一组微命令，控制完成下一步操作。完成若干步操作便实现了一条指令的功能，而实现了若干条指令的功能即完成了一段程

序的任务。

CPU 工作过程中所需的控制信号，既可以单独由组合逻辑（Combinational Logic）电路的方式来产生，也可以单独由微程序（Micro-program）的方式来产生，或者综合运用组合逻辑和微程序这两种方式来共同产生。因此有两种控制部件：组合逻辑控制器和微程序控制器，对应的两种控制方式分别为组合逻辑控制方式和微程序控制方式。在目前的微处理器领域中，这两种方式常被混合使用来产生各类控制信号。

4. 时序系统

不管是哪一种控制方式，要想在正确的时间产生所需的控制信号来控制 CPU 中各部件协调工作，必须有时间信号的支持。CPU 执行一条指令的过程是分步进行的，各步操作在时间上存在确定的联系，这就要求应有一组时间信号来作为各步的标志，如周期和节拍等。节拍是执行一步操作所需的时间，一个周期可能执行几步操作，所以可能包含几个节拍。一条指令在执行过程中，只有根据不同的周期、节拍等时间信号，才能在恰当的时间发出正确的微命令，控制执行部件完成相应的指令功能。

许多操作要求严格的定时控制，如在约定的时刻将数据送入某个寄存器，或者进行周期的转换，在规定的时刻结束当前周期的操作，并转入下一个周期。这些定时操作需要同步脉冲进行控制，在脉冲的上升沿或下降沿完成定时控制。

周期、节拍、脉冲等时间信号称为时序信号，产生时序信号的部件称为时序发生器或时序系统，它是由一个晶体振荡器和一组计数倍频逻辑组成的。晶体振荡器是一个脉冲源，能输出频率稳定的基准时钟脉冲，也叫外频。晶体振荡器一般位于主板之上，它也是计算机系统中其他部件，如主存和总线等部件的时序信号来源。CPU 的工作频率信号就是在外频信号的基础上，经过倍频电路将系统时钟频率放大以后得到的。

CPU 在工作时，各部件都要严格受控于统一的时钟信号，具体而言就是时钟周期的上升沿或者下降沿，所有部件的动作都要时钟信号去触发。因此，从控制模式来讲，CPU 采用的是一种同步控制（也叫同步定时）方式。实际上，不单是 CPU 采用同步控制方式，计算机系统其他部件也大多采用同步控制方式。

机器加电后，振荡器就开始振荡，但当 CPU 启动或停机时有可能与振荡器不同步，导致产生残缺的脉冲信号，就会使工作不可靠。因此需要设置一套启停控制逻辑，保证可靠地送出完整的时钟脉冲。启停控制逻辑在加电时还产生一个总清信号，或称为复位信号（RESET），对计算机中的有关部件进行初始化。

5. CPU 内部的数据通路结构

CPU 内部各部件之间需要传输信息，如寄存器将操作数送往 ALU 进行运算，ALU 将运算结果送入寄存器存放等。如何将这组部件连接起来，为信息传送提供通路呢？这就涉及 CPU 内部的数据通路结构，这是 CPU 组成的核心问题。不同的计算机，由于设计目标和设计方法不同，使 CPU 内部数据通路结构差异很大。我们主要讨论采用内部总线的数据通路结构，这种方式能使结构简单，较有规律并便于控制。

通常，在内部结构比较简单的 CPU 中只设置一组数据传输总线，用来连接 CPU 内的寄存器和 ALU 部件，在微处理器中常将这组总线称为 ALU 总线。在较复杂的 CPU 中，为了提高工作速度，可能设置几组数据总线，同时传输多个数据。有的 CPU 中包含了用于控制的存储逻辑以及内存管理所需的地址变换部件，因此除数据总线外，还设有专门传送地址信息的地址总线。下面

介绍几种典型的 CPU 数据通路结构。

(1) 单组内总线、分立寄存器结构

图 3-2 是一种早期的小型计算机所采用的 CPU 数据通路结构，这是一种最简单的通路结构，其特点是：采用分立寄存器，各寄存器有自己独立的输入、输出端口；用一组单向数据总线连接寄存器和 ALU，使 ALU 成为内部数据传送通路的中心。

在这种单向总线结构中，ALU 通过移位器只能向内部总线发送数据，不能直接从内总线接收数据。各寄存器能够从内总线上接收数据，但不能直接向内总线发送数据。如果一个寄存器要向另一个寄存器传输数据，则只能将数据送往 ALU，通过 ALU 来传输。因此，在 ALU 的输入端汇集了多个数据，需设置两个多路选择器，每次最多可以选择两个寄存器的内容，送入 ALU 进行运算；或者只选择一个寄存器的内容，经 ALU 送至另一寄存器。

ALU 既是运算处理部件，也是 CPU 内数据传送通路的中心，各寄存器的内容，不管是需要进行运算处理，还是只需简单的传输，都要通过 ALU 后再分配到目的寄存器。对数据来源的选择控制，集中在 ALU 输入端的多路选择器；对数据传输目的地的选择控制，则体现为寄存器同步打入脉冲的发送，使得对数据传输的控制变得比较简单、集中，这是该数据通路结构的主要优点。例如，要将寄存器 R_0 的内容送入寄存器 R_1 ，需要在 ALU 的输入端发微命令，使 R_0 通过一个输入选择器（如选择器 A）送入 ALU，同时封锁另一个输入选择器；ALU 的功能设置为“输出 A”；移位器的功能选择为“直接传送”，这样 R_0 的内容经选择器 A、ALU、移位器等被送到内总线上。当内总线上的数据稳定后，发输入到 R_1 的输入脉冲， R_0 的内容就被输入到 R_1 中，完成了寄存器到寄存器之间的数据传送。当然，这种分立寄存器结构使所需元器件和连接线增多，不利于集成度的提高。

(2) 单组内总线、集成寄存器结构

为了提高寄存器的集成度，寄存器组常采用小型半导体存储器结构，一个存储单元相当于一个寄存器，存储单元的位数即寄存器字长。图 3-3 就是一种采用集成寄存器结构的 CPU 数据通路结构。在这种结构中，用半导体随机存储器作为寄存器组，用一组双向数据总线连接寄存器与 ALU，并在 ALU 的输入端设置暂存器。

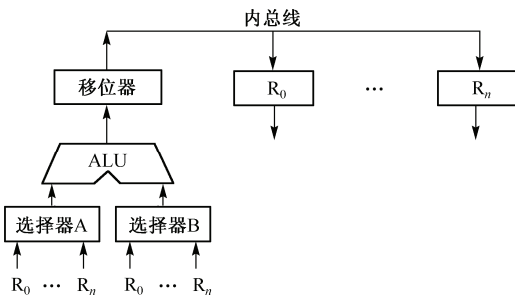


图 3-2 分立寄存器结构的 CPU 数据通路

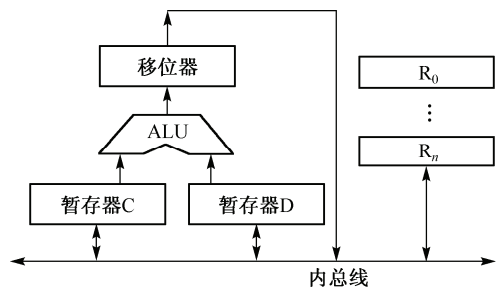


图 3-3 集成寄存器结构的 CPU 数据通路

图 3-3 所示模型采用双向数据总线，使 ALU 不仅可以向内总线输出运算结果，也可以从内总线上接收数据；各类寄存器也能直接从内总线接收数据和向内总线发送数据。这样，寄存器与 ALU 之间、寄存器与寄存器之间的数据传输都可以在这组内总线上进行，因而进一步简化了 CPU 内部数据传送通路的结构。

由于内总线每次只能提供一个操作数，而 ALU 本身又不具备暂存数据的能力，因此需要在 ALU 的输入端设置暂存器，暂存由内总线送来的数据。例如，要将寄存器 R_0 的内容与寄存器 R_1

的内容相加，可以先将 R_0 的内容经内总线送入暂存器 C，再将 R_1 的内容通过内总线送入暂存器 D，两个操作数准备好后送入 ALU 相加。此外，由于寄存器组采用的是单端口 RAM，每次只能访问一个寄存器单元，要在寄存器之间传输数据，就需要用暂存器作为中间暂存部件。例如，要将 R_0 的内容写入 R_1 ，可以先将 R_0 的内容送入某个暂存器，再由暂存器送至 R_1 。具体设计时，暂存器的数量、位置、传送方向等可以有多种变化。

(3) 多组内总线结构

以上两种 CPU 数据通路结构中都只有一组数据总线，其优点是结构简单、控制容易，但每个节拍内只能完成一种基本的数据通路操作，即将数据从一个来源地送往一个或多个目的地，这就使 CPU 的整体工作速率较低。而速率较高的 CPU 可能需要设置多组数据总线，一个节拍内可以并行地实现几种数据通路操作，即同时将多个数据从几个来源地分别送往各自的目的地。

现在的 CPU 结构往往更复杂，例如：设置有指令栈（或称指令队列），以便能预取若干条指令；有相应的操作数栈；有专用的 ROM（或称控制存储器）来存放微程序；有多个运算部件，用于存储管理的段地址运算部件和页地址运算部件；还有片内高速缓冲存储器 Cache 等。为了尽快执行指令，需要在 CPU 各类部件之间建立多种内部总线，传送数据、指令、地址等信息。本章 3.8 节中将具体介绍几种典型的 CPU 组成。

3.1.2 CPU 的工作原理

笼统地讲，CPU 的功能主要是执行程序和控制信息的输入输出。

1. CPU 的主要功能

① 处理指令。CPU 处理指令的含义是指控制程序中指令的执行顺序。程序中的各指令之间是有严格顺序的，必须严格按程序规定的顺序执行，才能保证计算机系统工作的正确性。

② 执行操作。一条指令完成的功能往往是由计算机中的各类工作部件执行一序列的操作来实现的。CPU 要根据指令的编码信息产生相应的操作控制信号，发给相应的部件，从而控制这些部件按指令的要求进行动作。

③ 控制时间。时间控制就是对各种操作实施时间上的定时。在一条指令的执行过程中，在什么时间做什么操作均应受到严格的控制，这样计算机系统才能有条不紊地工作。

④ 处理数据。数据处理是指对数据进行算术运算、逻辑运算及其他处理，如格式转换等。

总体来看，CPU 的工作过程就是从主存（或缓存）中读取指令，将指令放入指令寄存器（IR），然后对指令译码，把指令分解成一系列的微操作，再发出各种相应的控制命令，控制各功能部件执行相关的操作，从而完成一条指令的执行，实现对应的功能。

因此，CPU 的工作过程概括为 4 个阶段：

① 取指令。根据 PC 指定的主存地址，从主存（或缓存）中读取指令，放入指令寄存器 IR 中。

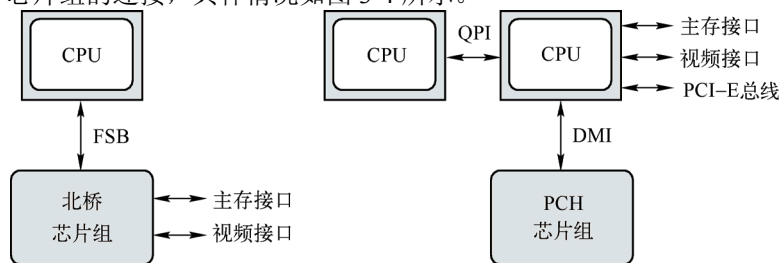
② 指令译码。CPU 根据 IR 中保存的指令，结合指令系统定义的规范，将指令分解成操作码、地址码等部分。操作码指示要进行哪种运算，地址码则指示如何得到操作数、结果如何保存，以及后续指令的地址如何形成等。

③ 指令执行。在指令的执行阶段中，主要完成操作数的读取、按指令操作码执行运算，暂存运算结果甚至形成后继指令地址等。

④ 后续阶段。主要完成将计算结果保存到主存（或缓存），有时还响应外部的某种请求，如响应外部的中断或者 DMA 请求等。

2. CPU 外部连接

现代计算机系统中，CPU 与外部的连接模式主要体现为两种，一种是与北桥芯片组的连接，另一种是与 PCH 芯片组的连接，具体情况如图 3-4 所示。



(a) 双桥布局下的 CPU 连接

(b) 单桥布局下的 CPU 连接

图 3-4 CPU 与外部的连接方式

图 3-4(a)表示经典的“南-北”桥布局中，CPU 直接通过 FSB（Front Side Bus，前端总线）与北桥芯片组的连接情况。这种模式下，因 CPU 没有集成内存控制器和显示核心，因此只能与北桥芯片组连接。图 3-4(b)表示的是单桥模式下，CPU 的外部连接情况。CPU 与其他 CPU 可以通过 QPI（Quick Path Interconnect）总线标准实现互连，并通过 DMI（Direct Media Interface）总线标准与 PCH 芯片组连接。此外，因这一代 CPU 一般都集成了内存控制器和显示核心，因此还可以直接与主存及视频接口相连，并且能提供若干路 PCI-E 标准的总线接口。

3. CPU 控制信息传送的方式

通俗地讲，可以把 CPU 看成是计算机系统的大脑。CPU 除了执行程序以外，还要承担主机与外围设备之间信息传送操作的控制任务。

总体上，CPU 常通过下列几种方式来控制“主机-外设”之间的信息传输：直接程序传输方式，程序中断方式，DMA 传输方式，IOP 方式（如通道）。直接程序传输方式全程需要 CPU 通过执行程序来实现输入输出操作的控制。程序中断方式一般需要 CPU 响应外部设备的随机请求，在响应随机请求以后，CPU 会暂停执行当前的程序，转向执行相应的中断服务，执行完成后再返回被暂停的程序继续执行。DMA 传输方式也需 CPU 响应随机请求，但 CPU 响应 DMA 请求以后，具体的输入输出操作由 DMA 控制器去控制，CPU 不需参与。IOP 方式下，CPU 启动 IOP（如通道）后，具体的输入、输出操作由通道控制器执行通道程序来完成，CPU 也不参与具体的输入、输出操作控制，且通道的传输效率比 DMA 的高。有关 CPU 控制“主机-外设”输入、输出操作的详细过程将在第 5 章中重点讲述。

3.1.3 CPU 的指令集类型

如前所述，CPU 是通过执行指令来完成某种运算或进行某种控制的。每款 CPU 在设计之初就应定义好了一系列与其硬件电路相适应的指令系统，如指令的格式、指令的类型、指令的数量以及指令涉及的各种寻址方式等。由此可见，指令集关系到了计算机系统的硬件结构和基本的硬件功能，因此它是 CPU 的一个重要指标。

从现阶段的主流 CPU 的指令架构来看，计算机主要有两种设计模式：复杂指令集计算机（Complex Instruction Set Computer，CISC）和精简指令集计算机（Reduced Instruction Set Computer，RISC）。

1. CISC

早期的计算机部件比较昂贵，CPU 的主频较低，运算速度慢。为了提高运算速度和扩展功能，

不得不将越来越多的复杂指令加入到指令系统中，以提高计算机的处理效率，这就逐步形成了 CISC 架构模式。

CISC 的特点是：指令系统庞大，指令功能复杂，指令格式多变，寻址方式也很多；绝大多数指令需要多个时钟周期才能完成；各种指令几乎都可以访问存储器；主要采用微程序控制方式；设置有少量专用寄存器；难以用优化编译技术生成高效的目标代码。

Intel 的 X86 系列（IA-32 架构）、AMD 和 VIA 等公司的 X-86 兼容系列及后来的 X86-64（或 AMD 64）架构系列处理器，基本上都属于 CISC 类型。

在 CISC 中，各种指令的使用频率悬殊很大，约有 20% 的指令高频使用（约占程序的 80%），其余 80% 的指令则较少使用（约占程序的 20%），这就是著名的“指令二 - 八”规律（1975，John Cocke，IBM 公司）。而且，为了实现大量的复杂指令，CPU 的控制逻辑会十分复杂且极不规整，从而限制了超大规模集成电路（VLSI）技术在处理器设计和生产中的应用。在 20 世纪 80 年代，飞速发展的半导体存储技术使很多之前需要复杂指令才能完成的功能，改用子程序来实现也能获得差不多的运算速度，这就使精简指令集成为了可能。

2. RISC

RISC 是一种执行较少类型计算机指令的微处理器，起源于 20 世纪 80 年代的 MIPS 主机（即 RISC 机），RISC 计算机中采用的微处理器统称 RISC 处理器。

相对于 CISC，RISC 不仅精简了指令系统，其指令格式更统一、指令种类也更少甚至寻址方式更简单，而且采用了“超标量和超流水线”结构，大大增加了 CPU 的并行处理能力，处理速度提高很多。RISC 指令集是高性能 CPU 的发展方向，在中高档服务器中普遍采用，且更适合高档服务器的操作系统，如 Windows 7/8 和 Linux 等。

在中高档服务器中采用 RISC 的 CPU 主要有：IBM 公司的 Power PC 处理器、SUN 公司的 SPARC 处理器、HP 公司的 PA-RISC 处理器、MIPS 公司的 MIPS 处理器和 Compaq 公司的 Alpha 处理器等。此外，一些低端处理器也属于 RISC 类型，比较常见的有 Acorn 公司发布的 ARM 和 Cortex 系列处理器。

除 RISC 和 CISC 之外，实际上还有显式并行指令计算机（Explicitly Parallel Instruction Computer, EPIC），至于它是否属于一种新的指令集类型，一直存在争议。EPIC 由 Intel 和 HP 公司于 1997 年联合提出，并于 2001 年在安腾（Itanium）处理器上首次采用，且安腾也是第一款基于 IA-64（X92）架构的处理器。

在 EPIC 的指令中，有 3 位是专门设置用来指示前后指令相关性的，如果前后指令之间无相关性，则这两条指令可以分别被分配到两个不同的 CPU 运算节点上同时处理，EPIC 还允许处理器根据编译器的调度来并行地执行指令，而不只是通过复杂的硬件电路。

3.1.4 CPU 的发展历程

计算机的发展历程总体上历经了 4 个发展阶段，每个阶段实际上都是由 CPU 技术的发展而推动的。每出现一种技术更先进的 CPU，都会带动计算机的组成部件随之发展，同时会促进计算机体系结构进一步优化，如存储器存取容量不断增大、存取速度不断提高，外围设备不断改进以及新的应用设备不断推陈出新。

在电子管、晶体管计算机时代，一般通过复杂的硬件逻辑来使计算机实现运算，那时的 CPU

集成度很低、体积庞大、功耗高、运算速度较低，也正是中小规模集成电路技术的发展（第三代计算机）促进了微处理器 CPU 的诞生。根据 CPU 的制造工艺、集成度、基本字长和指令功能的变化情况，可将 CPU 的发展历程概括为如下 8 个阶段。

（1）第一阶段：1946—1970

这个阶段的 CPU 主要由电子管或者较低集成度的晶体管构成，体积一般很庞大，功耗比较高，且运算速度一般很慢，适合处理 32 位定点数或者超过 40 位的浮点数四则运算，主要用于军事国防领域的计算机。在这个阶段中，CPU 还没有微处理器的概念。

代表性产品：国产 103 机的 CPU 及“功勋计算机”109 丙的 CPU。

（2）第二阶段：1971—1973

这个阶段是中小规模集成电路的初期，以 4 位和 8 位低档微处理器为主，主要特点是：采用 PMOS 半导体工艺，集成了约 4000 个晶体管，处理器架构和指令系统都比较简单，有 20 多条基本指令，基本指令周期为 20~50 μs ，主要用于一些简单控制。从本阶段开始，CPU 正式进入微处理器时代。

代表性产品：Intel 4004/8008。

（3）第三阶段：1974—1977

这个阶段以 8 位中高档微处理器为主，主要特点是：普遍采用 NMOS 半导体工艺，芯片的集成度大约为 16000 个晶体管，处理器的指令周期约 1~2 μs ，而且对应的指令系统逐渐完善，基本上具备中断和 DMA 等 I/O 控制功能。

代表性产品：Intel 8080/8085、Zilog Z80。

（4）第四阶段：1978—1984

这个阶段以 16 位微处理器为主，其特点是采用 HMOS 3 微米工艺，集成了约 20000~70000 个晶体管，指令周期约 0.5 μs ，指令系统更加丰富完善，支持多级中断和多种寻址方式，具备段式存储机构和硬件乘除等部件。

代表性产品：Intel 8086/8088/80286，Motorola M68000，Zilog Z8000 等。

（5）第五阶段：1985—1992

这个阶段以 32 位微处理器为主，其特点是采用 HMOS 或 CMOS 2 μm 工艺，集成度高达 100 万个晶体管，具有 32 位地址线和 32 位数据总线，指令周期约 0.16 μs 。1989 年，Intel 发布的 80486 首次突破 100 万晶体管的集成度，并集成了协处理器，是 80X86 系列的首款采用多重流水线技术的处理器。

代表性产品：Intel 80386/80486，Motorola M69030/68040 等。

（6）第六阶段：1993—2002

这个阶段还是以 32 位微处理器为主，典型产品是 Intel 的奔腾系列和与之兼容的 AMD K6、K7 系列微处理器芯片。

1993 年，Intel 公司发布了奔腾（Pentium）处理器即 586，内建 MMX（Multi Media eXtended，多媒体指令集），0.8 μm 工艺，外围电路采用芯片组方式。与 Pentium MMX 属于同一级别的有 AMD K6、Cyrix 6x86 MX 等。

1997 年，Intel 推出 Pentium II 处理器，SLOT1 架构，0.35 μm 工艺，SEC（Single Edge Contact）匣型封装，集成度约 750 万个晶体管。1998 年，Intel 推出了 Pentium II 的低端版本，即 Celeron（赛扬）系列，最大的差别就是取消了第二级缓存，后又在 Celeron 300A 中恢复。

1999 年，Intel 推出 Pentium III 处理器，在 Pentium II 的指令集中加入 70 个新指令，采用 SIMD

(Single Instruction Multiple Data) 技术, Socket 370 接口, 0.25 μm 半导体工艺, 集成度约为 950 万个晶体管。此外, 同级产品还有 Pentium III 的低端版本 Celeron 3 系, 0.13 μm 工艺制程。

2000 年, Intel 发布 Pentium 4 处理器, 采用 SSE2 指令集, 又新增 144 个指令, 集成了约 4200 万个晶体管, 采用 Socket 478 架构, 0.18 μm 半导体工艺, 256 KB 二级缓存, 主频 1.8~2.4 GHz。代表性产品是 Pentium 4 5X0 和 Pentium4 5X5 系列。

对于 32 位处理器, 2003 年, Intel 针对笔记本电脑推出了 Pentium M 处理器, 结合了 855 芯片组与 Intel PRO/Wireless 2100 网络接入技术, 成为 Intel Centrino (迅驰) 移动计算的最重要组成部分, 主频达 1.60 GHz, 并采用能耗最佳化的 400 MHz 系统总线、微处理作业融合以及专用堆栈管理器, 使处理器能快速执行指令集且功耗很低。

2002 年, 中科院计算所发布了我国第一款通用处理器“龙芯一号”, 32 位内核, MIPS III 指令集, 具有七级流水线、32 位整数单元和 64 位浮点单元, 主频 266 MHz; 2005 年, 发布“龙芯二号”, 0.18 μm 制程, 主频 1.0 GHz, 相当于 Pentium 4 的速度。

(7) 第七阶段: 2003—2004

这个阶段以 64 位的单核微处理器为主。2003 年, 对于 64 位微处理器, Intel 还是专注于 Itanium 处理器, 但 AMD 发布面向台式机的 64 位 Athlon 64 和 Athlon 64 FX 系列处理器, 采用 AMD-64 架构, 初始频率为 2.0 GHz, 0.13 μm 工艺, 集成度约 1.059 亿个晶体管; 同年, 苹果公司也推出一款 64 位的 Power PC 970 处理器 (G5)。

直到 2004 年, Intel 才发布了 EM64T 架构, 推出 64 位 Pentium 4 系列处理器, 采用 LGA 775 接口, 代表性的产品有 Pentium 4 5X1 和 5X6、Pentium 4 6XX 系列、Celeron D。同年, Freescale (飞思卡尔) 也推出了 64 位的 e700 core, VIA Technologies (威盛) 发布了 64 位的 Isaiah 处理器。

(8) 第八阶段: 2005—现在

这个阶段的 CPU 正式进入 64 位的多核处理器时代。

2005 年, Intel 推出双核处理器 Pentium EE (Extreme Edition)、Pentium D 和 Xeon 系列, 并推出了 945/955/965/975 芯片组来支持, 两者都采用 90 nm 工艺, 使用无引脚 LGA 775 接口标准, 有 755 个触点 (贴片电容), 通过与插槽内 775 根触针接触来传输信号。

随后, AMD 也推出了首款双核处理器 Athlon 64 X2, 集成 1 MB 二级缓存, 集成度高达 2.332 亿个晶体管。IBM 也推出 Power PC 970 MB 双核处理器。

2006 年, 进入 Core (酷睿) 架构时代, 采用 65 nm 工艺, 集成了 1.5 亿个晶体管, 原生双核共享 2MB L2, Socket 479 Core 接口。随后进入 Core 2 架构时代, 前端总线提升至 1333 MHz, L2 达 4 MB, LGA 775 接口, 增加的全新智能缓存技术提高了双核心乃至多核心处理器的工作效率, 代表性产品有 Core 2 Duo (双核) 和 Quad (四核) 系列。

随后 Core 架构发生了变革。2008 年, Intel 陆续发布了基于 Nahalem 微架构的第 1 代酷睿 Core i7/i5/i3 系列处理器产品。AMD 也于 2009 年开始, 陆续推出了 AMD Athlon II 系列多核处理器。2010 年, Intel 陆续发布了基于 Sandy Bridge 微架构的第 2 代酷睿 Core i7/i5/i3 系列处理器产品; 2012 年, Intel 正式发布了基于 Ivy Bridge 微架构的第 3 代酷睿 Core i7/i5/i3 系列处理器产品; 2013 年, Intel 甚至陆续推出基于 Haswell 微架构的第 4 代酷睿 Core i7/i5/i3 系列处理器产品。到目前, 陆续有 6 核、8 核、16 核甚至数十个内核的处理器问世, 并已开始各类计算机产品中应用。

我国首款多核处理器是中科院计算所 2009 年发布的“龙芯”3A (4 核, 65 nm 工艺, 主频为 1 GHz), 随后在 2012 年推出了“龙芯”3B-1500 处理器 (8 核, 28 nm 制程, 最高主频为 1.5 GHz)。此外, 国防科大专门为“天河一号” (2009 年 10 月) 研制了 FT-1000 (8 核 64 线程), 为“天河

二号”(2013年5月)研制了FT-1500(16核, SPARC V9架构, 40 nm工艺制程, 主频达1.8 GHz)。这些都是国产处理器的代表。

纵观CPU几十年的发展历程可以看出, 它的发展具有如下特点: 半导体制作工艺越来越精密, 从最初的几微米到现在的几十纳米, 芯片集成度呈几何级数提高; 位宽从早期的4位、8位、16位、32位处理器发展到现在的64位甚至128位处理器, 位宽增加了至少16倍; 从单核单线程发展到多核多线程, 处理器性能发生了翻天覆地的变化; 工作频率由早期的MHz发展到现在的GHz, 工作频率提高了1000倍左右; 指令集越来越丰富, 硬件架构越来越复杂, 硬件功能越来越强。

在通用处理器领域, Intel和AMD两大巨头无疑独领风骚; 在高性能计算领域, IBM和HP等公司的处理器尚能占有一席之地。国产处理器与国外处理器巨头发布的产品相比, 其技术水平差距较大, 基本处于技术引进和仿制阶段, 也仅有少数几款处理器在高性能计算领域有零星应用。无论是技术创新还是应用普及, 国产处理器都任重而道远。

3.2 指令系统

CPU的工作基本上体现为执行指令, 可以从两种角度来理解指令的含义。从程序的编制和执行角度, 用高级语言编制的程序经过编译, 转换为可由硬件直接识别并执行的程序形态, 即二进制的指令序列。每条指令能控制计算机实现一种操作, 因此指令中应规定操作的类型及操作数地址, 它们是产生控制信息的基础。从硬件角度, 在设计计算机时首先要确定其硬件能直接执行哪些操作, 既然每种操作都对应一条指令, 那么这些操作集合在一起就可以被看成是一个指令集。

所谓指令系统, 就是指计算机能执行的全部指令的集合, 可以看成是计算机硬件的语言系统, 也是软件、硬件的重要典型分界面。计算机的指令系统包含了若干条指令, 因此指令系统必须对指令的格式、功能、类型、数量以及相关的寻址方式等做出明确定义。本节将介绍指令系统所涉及的一些基本概念: 指令格式、寻址方式、指令类型等。

3.2.1 指令格式

1. 指令中的基本信息

在一条指令中通常包含以下信息:

① 操作码。指令应当给出操作的类型, 即要求计算机执行什么操作, 如加、减、乘、除等。为此, 指令中用若干位的编码去表明操作性质, 这小段编码叫做操作码。每条指令都有一个对应的操作码, 它也是区别不同指令的主要依据。

② 操作数或操作数的地址。参与运算操作的数据被称为操作数, 指令应当给出操作数的有关信息。在个别情况下, 指令中直接给出本次操作的操作数; 在大多数情况下, 指令只给出操作数存放处的地址, 如操作数所在寄存器的寄存器号, 或者操作数所在主存储器单元的地址码。由于多数情况下指令是给出操作数地址, 并指明CPU如何根据它们去寻找操作数, 所以一般认为指令中的基本信息是两部分: 操作码和地址码。

③ 存放运算结果的地址, 即运算完成后所得到的运算结果应当存放在何处。

④ 后继指令地址。为了求解一个问题, 往往需要一段程序, 最后成为一段可以执行的指令序列, 即若干条指令。因此, 在分析一条指令时应当想到, 它只是若干条指令中的一条, 当这条指令(称为现行指令)执行完后, 应到何处去读取下一条指令(后继指令)。存放后继指令的主存储器单元的地址码, 称为后继指令地址。后面将会说明, 后继指令地址多以隐含方式给出, 不

在指令中直接给出。

因此，大部分指令的基本格式如下，包含两类基本信息：

| | |
|--------|-------|
| 操作码 OP | 地址码 A |
|--------|-------|

如前所述，一条指令中可能包含几个地址码。进一步讨论，指令格式将涉及下述几方面：地址结构、操作码结构、指令长度。

2. 指令中的地址结构

指令的地址结构是指在指令中明确给出几个地址、给出哪些地址。在大多数指令中，地址信息所占的位数最多，因此地址结构是指令格式的一个重要问题。

如果在指令代码中明显地给出地址，如在指令中写明主存储器单元地址码或者寄存器号，则这种地址称为显地址。一条指令中给出几个显地址，就称为几地址指令。按照地址结构，指令可分为三地址指令、二地址指令、一地址指令、零地址指令。

如果地址是以隐含的方式约定，而指令中并不给出该地址码，则这种隐含约定的地址就称为隐地址。例如，事先约定操作数在某个寄存器中，或者约定操作数在堆栈之中（有关堆栈的概念后面再介绍）。为什么要采取隐含约定的方法呢？因为地址信息需占用指令的大部分位数，如果所有的地址信息都以显地址方式在指令中给出，势必使指令变得很长，导致程序所需占用的存储空间增大，读取和执行指令所需时间也会增加。为了解决前述问题，就需要简化指令的地址结构，最常使用的办法就是减少指令中“显地址”的数量，而减少指令中“显地址”数量的有效措施就是在指令中使用“隐地址”。

一条指令的操作所涉及的操作数数目与操作类型有关，如加、减运算涉及两个操作数，加 1、减 1 运算只涉及一个操作数，有些指令不涉及操作数。因此，从操作数的数量这一角度，可将指令分为双操作数指令、单操作数指令、无操作数指令。大多数指令所涉及的操作数不超过两个。

对于常规的双操作数运算，指令本应给出 4 个地址：操作数 1 存放地址、操作数 2 存放地址、运算结果存放地址、后继指令地址。如果这 4 个地址都以显地址的形式出现在指令中，这种指令就是四地址指令。但这种地址结构所需的指令位数太多，而且直接给定后继指令地址的方式反而使程序不能根据运算结果灵活转移，所以需要采用一些隐地址，减少指令中的显地址数，这称为简化地址结构。下面将分别举例说明。

按地址结构，实用指令可以分为以下几类。

(1) 三地址指令

指令格式：

| | | | |
|----|----|----|----|
| OP | A1 | A2 | A3 |
|----|----|----|----|

指令功能：(A1) OP (A2)→A3

(PC)+n→PC

由四地址指令简化为三地址指令，关键是让后继指令地址采用隐地址方式，这样一条双操作数指令就只需给出三个显地址。指令格式表明指令代码分成 4 段：OP 是操作码，表明这条指令做什么操作；A1 是操作数 1 所在地的地址，A2 是操作数 2 所在地的地址，A3 是运算结果的存放地址，它们可能是寄存器号，也可能是主存储器单元的地址码。

我们用一种寄存器级传送语句的形式来描述指令功能。(A1)表示按地址 A1 所读取的内容，即操作数 1。A1 是地址码，好比一所房屋的门牌号；(A1)是按 A1 地址读得的内容，它是数据，好比该房屋内的东西。

(A2)表示按地址 A2 所读取的内容，即操作数 2。寄存器传送级语句(A1) OP (A2)→A3 表明指

令功能是：分别按地址 A1 与地址 A2 读取操作数，按操作码 OP 进行运算操作，然后将结果存入地址 A3 所指定的主存单元或寄存器。例如，操作码 OP 规定做加法操作，这条指令的功能是(A1)+(A2)→A3。

为了以隐含方式给出后继指令地址，CPU 中设置了一个程序计数器 PC。PC 是一个兼有寄存器和计数器两种功能的部件，其中存放着读取指令的地址。一个基本程序段往往依次存放在主存储器的一段连续区域中，假定它的第一条指令共占几字节单元，第 1 字节存放在 1000H 单元中(地址码为十六进制数 1000)。则执行这段程序时，先将程序的初始地址 1000H 送入 PC；然后按 PC 存放的指令地址访问主存储器，从中读取指令；每读 1 字节，PC 的内容加 1，当读完第一条指令(共 n 字节)后，PC 内容一共加 n 。如果 CPU 在执行完第一条指令之后是顺序往下执行的，则 PC 现在的内容(加 n 以后)就是后继指令地址。如果第一条指令要求程序转移至 2000H 处，则将转移地址 2000H 送入 PC，仍按 PC 新内容读取后继指令。所以，程序计数器 PC 的作用就像一个指针，它指引 CPU 从何处去读取指令。PC 又被称为指令指针。

在解释了 PC 的设置及其作用之后，现在来看指令功能的第二部分，它用寄存器级传送语句描述为：(PC)+ n →PC。其含义是：如果现行指令占 n 字节存储单元，则读完本指令(称为现行指令)后，PC 内容共加 n ，使 PC 指向后继指令地址。

隐含约定为由 PC 提供指令地址，所以指令代码中不需给出后继指令地址，即后继指令地址是一种隐含地址。因此，对 PC 内容的增量计数操作也是隐含约定的，指令代码本身无需说明。

注意：从一个寄存器或一个存储单元读取指令或数据之后，寄存器(或存储单元)中原来存放的内容并不丢失，除非将新的内容写入。因此，上述三地址指令执行之后，A1 及 A2 地址的操作数并不丢失，可以再次使用；该指令本身也仍然保存在原来的位置，可以再次调用执行，所以程序可以多次执行。

(2) 二地址指令

指令格式：

| | | |
|----|----|----|
| OP | A1 | A2 |
|----|----|----|

指令功能：(A1) OP (A2)→A1

(PC)+ n →PC

在许多情况下，两个操作数运算后有一个不需要保留。例如，两数相乘时，部分积的累加和将代替原来的累加和，后者就不需要保留。又如两数相除，原来的余数将被新的余数取代，没有必要保留。因此，可以将运算结果存放在不需要保留的那个操作数的所在地。根据这一点，可以将某些三地址指令进一步简化为二地址指令。

由地址 A2 提供的操作数，在运算后仍保存在原处，称为源操作数，A2 称为源地址。由地址 A1 提供的操作数，在运算后不再保留，该地址改用来存放运算结果。因为 A1 最终是存放运算结果的目的地，所以一开始由 A1 提供的操作数被称为目的操作数。相应地，指令的基本功能被描述为(A1) OP (A2)→A1，即分别按地址 A1 与地址 A2 读取操作数，按操作码 OP 进行运算操作，然后将结果存入地址 A1 所指定的主存单元或寄存器。后继指令地址的产生方法与三地址指令相同，不再赘述。

(3) 一地址指令

指令格式：

| | |
|----|---|
| OP | A |
|----|---|

一地址指令有两种常见的形态，可以根据操作码的含义确定它究竟是哪一种。下面分别介绍这两种一地址指令的隐地址方式。

① 只有目的操作数的单操作数指令。有些指令只需一个操作数，称为单操作数指令。例如，对某操作数加 1、减 1、求反、求补等指令，按地址 A 读取操作数，进行操作码 OP 要求的操作，运算结果存回原地址 A。如前所述，这样的操作数被称为目的操作数。

指令功能： $OP(A) \rightarrow A$
 $(PC)+n \rightarrow PC$

② 隐含约定目的地的双操作数指令。如果操作码含义是加、减、乘、除、与、或、异或等，说明该指令是双操作数指令。按指令给出的源地址 A 可读取源操作数，那另一个操作数呢？在 CPU 中一般设置有一个被称为累加器 AC 的寄存器，指令可以隐含约定另一个操作数（即目的操作数）由 AC 提供，运算结果也将存放在 AC 中。

指令功能： $(AC) OP(A) \rightarrow AC$
 $(PC)+n \rightarrow PC$

可见，一地址指令不仅可用于处理单操作数运算，也可用于处理双操作数运算，这是采用隐地址以简化地址结构的又一例子。

（4）零地址指令

指令格式：

| |
|----|
| OP |
|----|

如果指令中只给出操作码而没有显地址，则这种指令被称为零地址指令。有几种情况可能使用零地址指令。

① 不需要操作数的指令。例如，有一种空操作指令，它本身没有实质性运算操作，执行这种指令的目的就是消耗时间以达到延时的目的；又如，停机指令当然不需要操作数。

② 该指令是一条单操作数指令，并隐含约定操作数在累加器 AC 中，即对 AC 中的内容进行 OP 指定的运算操作。

指令功能： $OP(AC) \rightarrow AC$

③ 对堆栈栈顶单元中的数据进行操作。后面即将介绍，堆栈是一种按“后进先出”顺序进行存取的存储组织，每次存取的对象是栈顶单元，该单元是浮动的，由一个被称为堆栈指针的寄存器 SP 给出栈顶单元地址。因此，对堆栈的操作可以只给出操作码，隐含约定由 SP 提供地址。在学习了有关堆栈的知识后，大家就会明白，对堆栈操作的零地址指令既可以用于单操作数运算，也可以用于双操作数运算。

在上述分析中体现了一条思路：采用隐地址（隐含约定）可以简化指令的地址结构，即减少指令中的显地址数。例如，用隐地址方式给出后继指令地址、将提供操作数的地址与存放运算结果的目的地址统一为一个、隐含约定操作数在累加器 AC 之中或在堆栈中等。

大家可能要问，究竟是显地址数多好还是显地址数少好？应该说各有利弊。显地址数多，则指令长，所需存储空间大，读取时间长，但地址数多则使用较灵活。显地址数少，则指令短，所需存储空间小，读取时间短，但使用隐地址的方式会对地址选择带来一定限制。就目前情况看，设计者往往采取折中的办法，因此二地址指令与一地址指令使用频率较高。

3. 操作码结构

操作码的位数决定了操作类型的多少，位数越多所能表示的操作种类也就越多。例如，操作码有 8 位，则该指令系统最多可以有 256 种指令。但如果指令字长有限，地址部分的位数和操作码的位数就会相互制约，即如果地址部分所占位数较多，则允许操作码占用的位数就会相应地减少，从而限制了指令的种类数目。所以在操作码结构设计上有一些不同的方法。

(1) 固定长度操作码

如果指令较长（位数多），或者是采用可变字长指令格式，则往往采用固定长度操作码，即操作码位数一定且位置固定。现在计算机较多采用这种格式，最常见的指令格式是：一条指令由几字节组成，其中第 1 字节（8 位）表示操作码。

由于操作码的位数一定且位置固定，因此读取与识别指令都比较方便。例如，所读得的第 1 字节指令代码是操作码，就知道该指令是单操作数指令还是双操作数指令，以及相应的地址信息组织方法。此外，分析操作码含义所用的译码电路也比较简单。

(2) 可变长度操作码（扩展操作码）

为了提高指令的读取与执行速度，往往需要限制指令的字长。而要想在指令字长有限的前提下仍能保持比较丰富的指令种类，一种方法是采取可变长度操作码。这种方法是：当指令中的地址部分位数较多时，让操作码的位数少些；当指令的地址部分位数减少时（例如现在是一地址指令，因而地址位数相应减少），可让操作码的位数增多，以增加指令种类，这称为扩展操作码。当然，这会增加识别操作码的难度。

【例 3-1】 假设某指令系统的指令字长 16 位，最多可给出三个地址段 X、Y、Z，每个地址字段占 4 位（当然这只是一种示意性的假设，实际指令中每个地址字段不只 4 位）。试给出一种扩展操作码的方案。

图 3-5 给出了一种扩展操作码的原理性方案。

| 15 | 12 11 | 8 7 | 4 3 | 0 | 指令格式 |
|------|-------|------|------|---|-----------|
| 0000 | X | Y | Z | | 15 条三地址指令 |
| ... | ... | ... | ... | | |
| 1110 | X | Y | Z | | |
| 1111 | 0000 | Y | Z | | 14 条二地址指令 |
| ... | ... | ... | ... | | |
| 1111 | 1101 | Y | Z | | |
| 1111 | 1110 | 0000 | Z | | 31 条单地址指令 |
| ... | ... | ... | ... | | |
| 1111 | 1110 | 1111 | Z | | |
| 1111 | 1111 | 0000 | Z | | |
| ... | ... | ... | ... | | |
| 1111 | 1111 | 1110 | Z | | |
| 1111 | 1111 | 1111 | 0000 | | |
| ... | ... | ... | ... | | 16 条零地址指令 |
| 1111 | 1111 | 1111 | 1111 | | |

图 3-5 扩展操作码示意图

对于三地址指令，三个地址共占 12 位；操作码 4 位，如果全部用来表示操作功能，可以表示 $2^4=16$ 种，现在只取其中的 15 种组合 0000~1110，可分别表示 15 条三地址指令，留下 1111 作为扩展操作码标志。对于二地址指令，两个地址共占 8 位，尚余 8 位，在高 4 位（第 15~12 位）为 1111 的扩展标志指示下，将第 11~8 位扩展为操作码；如果只取其中 14 种组合表示二地址指令，即 11110000~11111101，则留下了两种组合 11111110、11111111 可作为扩展操作码标志。对于单地址指令，地址段只占 4 位，又可将第 7~4 位扩展为操作码。由于在设计二地址指令时留下了两种扩展标志，加上第 7~4 位的编码，可得 $2 \times 2^4=32$ 种组合。现在只取其中 31 种组合表示单地址指令，即 111111100000~111111111110，留下一组组合 111111111111 作为扩展标志。对

于零地址指令，在第 15~4 位为全 1 的扩展标志指引下，可有 16 种零地址指令（一般机器不需要这么多条零地址指令）。

图 3-5 给出的只是一种原理性的分配方案，实际机器会在细节上有一些变化。在这个例子中，我们有意为二地址指令只留下一种扩展标志，相应地只有 15 条三地址指令；而为单地址指令留下两种扩展标志，相应地只有 14 条二地址指令，而单地址指令可有 31 条。

(3) 单功能型或复合型操作码

为了能快速识别并执行操作码，多数指令采用单功能型操作码，让操作码只表示一种操作含义，如相加或相减。因为指令字长有限，有的计算机指令条数有限，为了使一条指令能表示更多的操作信息，采用复合型操作码，将操作码分为几部分，让它们的组合使操作含义比较丰富。

4. 指令字长

前面的讨论中已经多次涉及指令字长问题。指令字的位数越多，所能表示的操作信息和地址信息就越多，可使指令功能丰富。但位数多，则指令所占存储空间就多，相应地，读取指令的时间增长，而且指令越复杂则执行时间就越长。反之，指令字长固定、格式简单，则读取与执行时间就短。因此，在指令设计上出现两种截然相反的思路，一种是让指令功能尽可能丰富，指令数量也尽可能多，称为复杂指令系统计算机（CISC）；另一种是只选取简单且常用的指令，称为精简指令系统计算机（RISC）。与之对应，在指令字长的设计方面也出现了两种不同的思路。

图 3-6 给出了两种指令长度的设计思路，分别是固定字长和变字长指令。

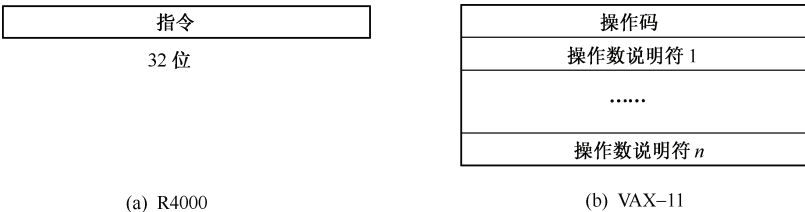


图 3-6 指令的长度类型举例

(1) 变字长指令

在传统的大、中、小型计算机中及常用微型计算机中，采用变字长指令格式，不同的指令可以有不同的字长，需长则长，能短则短。但因为主存储器一般按字节编址，即以字节（8 位）为基本单位，所以指令字长多为字节的整数倍，如单字节、双字节、三字节等。这种指令格式适于表示复杂指令系统。那么，CPU 怎么知道这条指令有多少字节呢？一般的做法是：将操作码放在指令的第 1 字节，读出操作码后就可以马上判定，这是一条单操作数指令还是一条双操作数指令，或者是零地址指令，从而知道后面还应该读取几字节指令代码。当然，在采取预取指令的技术时，这个问题会稍微复杂一些。

(2) 固定字长指令

在早期的小型计算机中曾经广泛采用固定字长指令，如一条指令 16 位。后来逐渐转化为变字长指令，以获得功能丰富的指令系统。但从 20 世纪 70 年代开始，又出现一种趋势，即采取精简指令系统以提高执行速度，相应地又恢复采用固定字长指令格式。随着一整套的精简指令系统技术（RISC 技术）逐渐成熟，RISC 技术已经成为一种非常重要的发展趋势，广泛用于工作站一类的高档微机，或采用众多的 RISC 处理器构成大规模并行处理阵列，对 PC（个人计算机）的发展趋势也产生影响。

3.2.2 寻址方式

所谓寻址，就是指产生操作数的有效地址，因此将产生操作数有效地址的方式称为寻址方式。CPU 根据指令约定的寻址方式对地址字段的有关信息做出解释，以找到操作数。有的指令通过操作码的含义隐含约定采用何种寻址方式；有的指令则设置专门的寻址方式编码字段，以说明采用何种寻址方式。如果是双操作数指令或数据传送指令，则指令涉及多个地址，各有自己的寻址方式，不一定相同，也就是说，一条指令之中可以有多种寻址方式。虽然寻址方式的基本含义是针对操作数的寻找，但程序转移指令需要提供转移地址，这与提供操作数地址的方法并无区别，因此可归入寻址方式的范畴并讨论。

一个指令系统具有哪几种寻址方式，即地址以什么方式给出，如何为编程提供方便和灵活性，这不仅是设计指令系统的关键，也是初学者理解一个指令系统的难点所在。因此，大家在学习本节内容时，要从众多的寻址方式中归纳出一条清晰的思路。

首先，我们想一想，指令要调用的操作数可能存放在什么地方？有以下几种可能。

① 操作数就包含在该指令之中，或紧跟着该指令。此时需要由指令直接给出操作数。

② 操作数在 CPU 的某个寄存器之中。例如，已由主存储器将操作数读入 CPU，或者运算处理的中间结果，因而它们在 CPU 的寄存器中。此时指令中应给出寄存器号。

③ 操作数在主存储器中，指令应以某种方式给出主存单元地址码。这里还可以分为几种情况：有的是对单个操作数进行处理，有的是对一个连续的数组或是对数组中的某个元素进行处理，有的是对一个表格或者对表格中的某个元素进行处理等。相应地，需要采取不同的寻址方式。

④ 操作数在堆栈区中。可以隐含约定由堆栈指针 SP 提供地址。

⑤ 操作数在某个 I/O 接口的寄存器之中。注意，这不是 CPU 中的寄存器。一般采取两种办法对 I/O 接口中的寄存器进行编址：一种是单独编址，指令中需提供 I/O 端口地址；另一种是与主存单元统一编址，指令中需提供总线地址。这些问题以后还要专门讨论。

然后，我们再想一想，为什么需要多种寻址方式供编程者选择呢？沿着从简到繁的思路，大致可将众多的寻址方式归纳为以下四大类。

① 立即寻址。在读取指令时就从指令之中获得了操作数。

② 直接寻址类。直接给出主存地址或寄存器号，以读取操作数。

③ 间接寻址类。先从某寄存器中或主存中读取地址，再按这个地址访问主存以读取操作数。间接寻址的目的是：在使用同一条指令的前提下，使操作数地址可以变化，从而增加编程的灵活性，使一条指令甚至是一段指令能被重复利用。

④ 变址类。指令给出的是形式地址（不是最后地址），经过某种计算（如相加、相减、高低位地址拼接等）才获得有效地址，据此访问主存储器，以读取操作数。采取变址计算的目的是使程序能更有效地适应各种需要，如对数组、表格、链等数据结构的访问，以及程序转移、存储管理、程序重定位等。

因此，尽管各种计算机的寻址方式甚多，在同一系列的计算机指令系统中往往有好几种寻址方式，而在不同系列之间更是既有大体相同之处，也有各具特色之处，它们对寻址方式的分类和命名也有各自的规定，但几乎都是以上述四类（立即、直接、间址、变址）为最基本的寻址方式，其他则是它们的变型或组合。沿着上述思路去学习，可以更好地理解各种寻址方式的含义与实现流程。

注意：对 CPU 指令读取与执行流程的理解，是本课的重点之一，而对各种寻址方式的理解则是掌握指令流程的关键。请大家务必注意。

下面分别介绍几种常见的寻址方式，请注意各种寻址方式的物理含义、寻址过程，以及各自的适用范围和具体的使用方法。

1. 立即寻址

由指令直接给出操作数，在取出指令的同时就取出了可以立即使用的操作数，这样的数被称为立即数（常数），这种寻址方式被称为立即寻址方式。在用助记符描述的指令语句中，立即寻址方式的助记符常用 I 表示，通常用于为程序提供常数或某种初始值。虽然立即寻址方式能快速获得操作数，但在多数场合中程序所处理的数据是变量，因此立即寻址方式的适用范围十分有限。

指令给出的操作数是怎样存放的呢？一般有两种方式。一种是让操作数占据指令中用来存放地址的位置，即操作数直接包含在指令的地址段中。这种方式下操作数就在指令之中，如图 3-7(a)所示，在取出指令的同时也就获得了操作数。另一种方式是将操作数存放在指令之后，即在读取指令之后再从紧随其后的存储单元（一个或数个单元）中读取操作数。这种情况可称为：操作数紧跟在指令之后，如图 3-7(b)所示。其实这两种方法并无实质区别，只是对指令所含内容的说法有所不同而已，如果将图 3-7(b)情况视为一条多字节的指令，则操作数就可视为指令的一部分，那么图 3-7(b)就跟图 3-7(a)没有实质区别了。

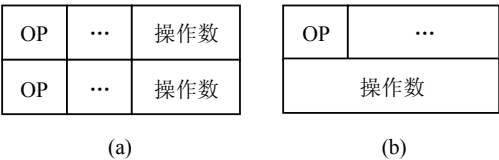


图 3-7 立即寻址方式示意图

2. 直接寻址

在指令中直接给出操作数的有效地址，根据该地址可从主存储器中读取或写入操作数，这种方式就称为直接寻址方式。由于这个地址就是最终读取操作数的有效地址，不再进行任何地址转换操作，故又将直接寻址称为绝对寻址。广义的地址既可能是主存地址也可能是寄存器号，但习惯上“直接寻址”方式中直接给出的操作数地址指的是主存地址。

在指令语句中，直接寻址方式的助记符常用(A)表示。例如，指令 INC (A)表示将地址为 A 的主存单元内容加 1。

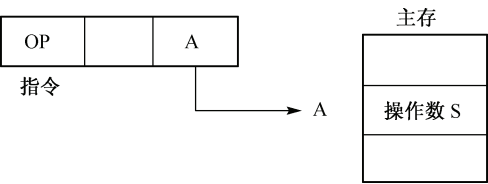


图 3-8 绝对寻址方式示意图

我们再从图例的形式解释直接寻址方式的含义，如图 3-8 所示。为了简化细节以突出基本原理，假定主存储器是按字编址的，一个操作数占一个主存单元，操作数本身为 S，主存单元的地址为 A；指令占一个字，包含了操作码 OP 和地址 A。

【例 3-2】若主存储器数据区的地址与数据之间对应关系如下，指令给出地址码 A=2000H，按直接寻址方式读取操作数。

| 地址 | 数据 |
|-------|-------|
| 1000H | 1A00H |
| 2000H | 1B00H |
| 3000H | 1C00H |

按照直接寻址方式的定义，所读得的操作数是 1B00H。

我们还可利用下面的形式描述直接寻址的过程，其中 M 是主存储器 Memory 的缩写，M 表示

访问主存储器 M。

操作数地址 \xrightarrow{M} 操作数

直接寻址方式的优点是简单直观，且便于用硬件实现，适用于寻找地址固定的操作数，但这种寻址方式也存在两点不足：

① 有效地址是指令的一部分，不能随程序的需要而动态地改变，因而该指令只能访问某个固定的主存单元。例如，例 3-2 中的那条指令就只能访问 2000H 单元，如果想要在多处重复使用这条指令，就会受到一定限制。

② 指令中需要给出全字长的地址码，由于地址码在指令中所占的位数较多，这会导致指令字的长度变得很长。

3. 寄存器直接寻址

寄存器直接寻址（简称寄存器寻址）方式：在指令中直接给出寄存器号，操作数实际存储在指定编号的寄存器中。CPU 中有寄存器若干，其中一些可编程访问，也称为通用寄存器。在设计时，要为寄存器分配不同的编号代码，如 R_0 —000、 R_1 —001 等。如图 3-9 所示，指令中给出的寄存器编号是 R_0 （代码为 000），故在 R_0 中可直接读取操作数 S。在指令中，寄存器寻址方式的助记符常用 R 表示，如指令语句 INC R_0 就表示将 R_0 中的内容加 1。

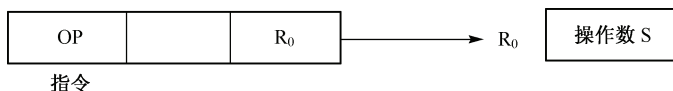


图 3-9 寄存器直接寻址方式示意图

【例 3-3】 若 CPU 中寄存器内容如下，现指令中给出寄存器号为 010（即 R_2 ），按寄存器直接寻址方式读取操作数。

| | | | |
|-------|-------|-------|-------|
| R_0 | 1000H | R_2 | 3A00H |
| R_1 | 2000H | R_3 | 3C00H |

按寄存器直接寻址方式定义，读得的操作数 $S = 3A00H$ 。

寻址过程：寄存器号 \xrightarrow{R} 操作数

寄存器寻址也是一种“直接”寻址，不过它是按寄存器号去访问寄存器的，与绝对寻址方式不同。寄存器直接寻址方式有以下两个主要优点，因而应用广泛：

① 从 CPU 的寄存器之中读取操作数要比访问主存快得多，所需时间大约是从主存中读数时间的几分之一到几十分之一。因此，在 CPU 中设置足够多的寄存器，以尽可能多地在寄存器之间进行运算操作，已成为提高工作速度的重要措施之一。

② 由于寄存器数远少于主存储器的单元数，所以指令中存放寄存器号的字段位数就大大少于存放主存地址所需的位数。采用寄存器直接寻址方式或其他以寄存器为基础的寻址方式（如寄存器间址方式），可以大大减少指令中地址位的宽度，能有效地缩短指令长度。这也使读取指令的时间减少，提高了工作速度。

注意：减少指令中地址数目与减少一个地址的位数是两个不同的概念。采用隐地址可以减少指令中地址的数目，采用寄存器寻址方式、寄存器间址方式可以使指令中为给出一个地址所需的存储位数减少。

4. 主存间接寻址

先定义几个术语。若操作数存放在某个主存单元中，则该主存单元的地址被称为操作数地址。

若操作数地址又存放在另一主存单元中（不是由指令直接给出），则该主存单元被称为间址单元，间址单元本身的地址被称为操作数地址的地址，即操作数的间接地址。

主存间接寻址方式的定义是：指令中给出主存间址单元地址（间接地址，操作数地址的地址），按照该地址访问主存中的间址单元，从中读取操作数地址，按操作数地址再次访问主存，然后从相应单元中读取或写入操作数。主存间接寻址方式的助记符常用@表示。例如，指令 `INC @A` 表示：间址单元地址码为 A，根据 A 访问主存并读取操作数的有效地址，再按这个有效地址访问主存并读取操作数，最后操作数加 1。

如图 3-10 所示，指令中给出的间接地址是 A1，根据 A1 访问主存中的间址单元，从中读取到操作数的有效地址 A2，再按 A2 访问主存，读取操作数 S。

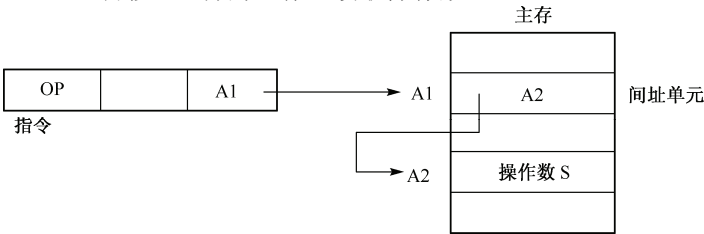


图 3-10 主存间接寻址方式示意图

将寻址过程归纳起来，可以被简洁地描述为：

间址单元地址 \xrightarrow{M} 操作数地址 \xrightarrow{M} 操作数

【例 3-4】 若主存储器数据区的地址与单元内容之间对应关系如下，指令给出的主存间接地址 A = 2000H，按主存间接寻址方式读取操作数。

| 地址 | 存储内容 |
|-------|-------|
| 1000H | 4000H |
| 2000H | 3000H |
| 3000H | AC00H |

按照主存间接寻址方式的定义，指令给出的间接地址是 2000H，据此访问主存，从中读取到操作数的有效地址 3000H，再按此地址再次访问主存，读取到操作数 S=AC00H。

采用主存间接寻址方式可将间址单元当成一个读取或写入操作数的地址指针，它指示操作数在主存中的位置。通过修改指针（即间址单元的内容），同一条指令就可以用来在不同时间访问不同的存储单元。虽然在同一条指令中所指定的间址单元的地址是不变的，但同一个间址单元的内容即操作数的地址却是可以变化和修改的，故使用同一条指令并通过不断修改间址单元的内容就能实现对不同操作数的访问。

这种寻址方式有力地支持了程序的循环操作，也能实现程序的共享。例如，在一个求平方根的程序中，基本操作指令重复使用，而各次运算的操作数随指针修改而变。又如，在分时多道程序工作方式中，几个用户可通过间址方式共享某段子程序，它们的编程工作是独立进行的，因而各自使用的间址单元不同。但如果这些间址单元的内容都是同一个公共子程序的入口地址，则它们就可以通过间址方式获得转移地址，从而转向该子程序，实现对公共子程序的共享。再如，在转向子程序时，可将返回地址存放在某个约定的间址单元。在返回主程序时，先执行一条返回指令，以间址方式从间址单元取出返回地址，据此返回主程序。对于一些公共子程序，它们使用的是同一条返回指令，且指向的间址单元也相同，但不同调用者（即主程序）在调用时填入的返回地址不同，因而返回时并不会引起程序混乱。

可见，这种通过主存间接寻址产生有效地址的方法为编程提供了很大灵活性，但它会增加访问的次数，因而会降低 CPU 的工作速度。此外，指令中给出间址单元的地址，还会使指令的长度增加。早期的一些计算机中还设计了多重间址，现在已很少用。

5. 寄存器间接寻址

寄存器间接寻址（简称寄存器间址）方式的定义是：指令中给出寄存器号，指定的寄存器中存放的是操作数的有效地址，按照该有效地址访问主存，读取或写入操作数。寄存器间接寻址方式的助记符常用(R)表示。例如，指令 INC (R₀)表示：将寄存器 R₀ 的内容作为有效地址，从主存中读取操作数，再将操作数加 1。

如图 3-11 所示，指令中在地址段给出寄存器号是 R，从该寄存器中读到的是操作数的有效地址 A，再按 A 访问主存，从相应的主存单元中读取到操作数 S。

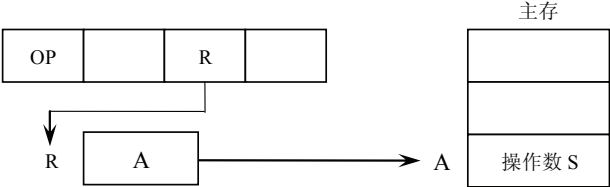


图 3-11 寄存器间接寻址方式示意图

寄存器间接寻址方式的过程可以简洁地描述为：

寄存器号 \xrightarrow{R} 操作数地址 \xrightarrow{M} 操作数

【例 3-5】 下面分别列出寄存器中存放内容，以及主存储器数据区的地址与单元内容之间的对应关系。若指令中给出寄存器号为 001，按寄存器间址方式读取操作数。

| | | | |
|--------------------|-------|------------|-------|
| 寄存器：R ₀ | 1000H | 主存单元：1000H | 3A00H |
| R ₁ | 2000H | 2000H | 2C00H |
| R ₂ | 3000H | 3000H | 3B00H |

按寄存器间接寻址方式的定义，指令中给出的寄存器号为 R₁，从中读得操作数的有效地址 2000H，据此访问主存，最后读到的操作数是 2C00H。

采用寄存器间接寻址方式，可以选取某通用寄存器作为地址指针，它指向操作数在主存中的存储位置。修改寄存器的内容，可使同一指令在不同时间访问不同的主存单元，提供了编程的灵活性，因此它也具有主存间接寻址方式的优点，如可以方便程序循环执行。但与主存间接寻址方式相比，寄存器间接寻址方式还具有两个显著优点：

- ① 寄存器间接寻址方式比主存间接寻址方式少访问一次主存，且由寄存器提供有效地址及修改寄存器内容，比从主存中读取有效地址及修改主存单元内容要快很多，因此寄存器间接寻址方式的执行速度较快。在编程中让寄存器充当地址指针已成为一项基本策略，这点请大家留意。
- ② 指令中给出的寄存器号位数比主存的全地址码位数少很多，且寄存器的宽度可以设计得很大，足够容纳全字长的地址码。如例 3-5 中，寄存器号只需 3 位，而对应寄存器可以容纳 16 位长度的地址码。因此，采用寄存器间接寻址方式也能减少指令中地址字段的位数。

为了提高 CPU 的工作速度，现在有些计算机不再采用主存间接寻址方式，但寄存器直接寻址方式和寄存器间接寻址方式是所有计算机几乎都具备的两种寻址方式。

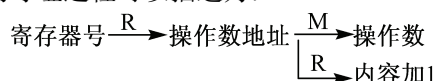
在程序中，有时需要对主存中的一个连续存储区域（如数组）进行操作，此时每读出一个操作数就需要执行一次指针修改，如加 1 或者减 1，以实现存储单元的连续访问。为适应这种应用

需求，寄存器间接寻址方式还派生出了两种变型：

(1) 自增型寄存器间址

指令中给出寄存器号，指定的寄存器中存放着操作数的有效地址，从寄存器读出操作数地址后，寄存器内容自动加 1，并按照有效地址从主存中读取操作数。自增型寄存器间接寻址方式常用助记符(R)+表示，如指令 INC (R₀)⁺。与标准的寄存器间接寻址方式相比，助记符中多了一个“+”，且位于(R)-之后，这就形象地表示：先操作（从寄存器中读取操作数的有效地址，按此地址访问主存，读取或写入操作数），后修改（寄存器的内容加 1）。

自增型寄存器间址方式的寻址过程可以描述为：



【例 3-6】 下面列出了寄存器内容，以及主存数据区的地址与单元内容。若指令中给出的寄存器号为 010，按自增型寄存器间址方式读取操作数，并修改寄存器内容。

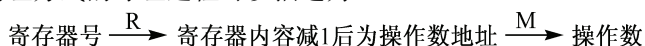
| | | | |
|--------------------|-------|------------|-------|
| 寄存器：R ₀ | 1000H | 主存单元：3000H | A300H |
| R ₁ | 2000H | 3001H | BC00H |
| R ₂ | 3000H | | |

按自增型寄存器间址方式的定义，指定的寄存器为 R₂，从寄存器中读得操作数有效地址 3000H 后，寄存器的内容自动加 1，即 R₂ 的内容被修改为 3001H，再按地址 3000H（注意，不是 3001H）访问主存，读得操作数 A300H。如此反复，通过重复执行这同一条指令就可以沿着地址码增加的方向，访问从 3000H 单元开始的一段连续数据存储区。

(2) 自减型寄存器间址

指令中给出寄存器号，指定的寄存器内容先减 1 以后再被读取出来作为操作数的有效地址，然后按此地址访问主存，从相应主存单元中读取操作数。自减型寄存器间址方式常用助记符-(R)表示，如指令 INC -(R₀)。与标准的寄存器间接寻址方式相比，助记符中仅多了一个“-”，但是位于(R)-之前，这形象地表示：先修改（寄存器内容减 1），后操作（把修改后的寄存器内容读出来作为操作数的有效地址去访问主存）。

自减型寄存器间址方式的寻址过程可以描述为：



【例 3-7】 下面分别列出寄存器内容，以及主存数据区的地址和单元内容。若指令中给出寄存器号为 010，按自减型寄存器间址方式读取操作数。

| | | | |
|--------------------|-------|-------------|-------|
| 寄存器：R ₀ | 1000H | 主存单元：2FFE H | A300H |
| R ₁ | 2000H | 2FFF H | 27FAH |
| R ₂ | 3000H | 3000H | BC00H |

按照自减型寄存器间址方式的定义，指定的寄存器为 R₂，先将 R₂ 的内容（3000H）减 1，R₂ 内容变为 2FFFH，然后读出 R₂ 的内容作为操作数的有效地址 2FFFH，据此访问主存，最后读得的操作数为 27FAH。如此反复，通过重复执行这同一条指令，就可以访问从地址 2FFFH 开始，沿地址码减小方向的一个连续数据存储区。

大家可能要问，为什么自增型寄存器间址方式(R)+一般设计成“先操作、后修改”，而自减型寄存器间址方式-(R)却设计成“先修改、后操作”呢？当学习了堆栈的压栈与出栈操作过程以后，大家就会明白这样设计的目的。

6. 变址寻址

变址寻址方式是另一种广泛应用的重要寻址方式，在几乎所有的计算机中都设置了这种寻址方式。如前所述，主存间址方式和寄存器间址方式是通过多层读取来提供地址的可变性的，而变址方式则是通过地址计算来使有效地址的形成更加灵活多变。

变址寻址方式的定义是：指令中分别给出一个寄存器号和一个形式地址，寄存器中的内容作为偏移量，形式地址作为基准地址，将基准地址和偏移量相加得到操作数的有效地址，再按此地址访问主存，从相应的主存单元中读取操作数，或把操作数写入此主存单元。变址方式常用助记符 X(R)表示。例如，指令 INC X(R₀)表示以寄存器 R₀ 的内容为偏移量，以指令后邻单元存储的形式地址 D 为基准地址，两者相加形成操作数的有效地址。注意：这里的“X”仅是变址寻址方式的助记符，不代表数值或地址码。

如图 3-12 所示，为了获得操作数的有效地址，指令给出了两个信息：一个是形式地址 D（不是有效地址，只是一个形式上的地址）；另一个是变址寄存器号 R_x（可指定为某个通用寄存器），R_x 中存储的内容 N 为偏移量。按变址寻址方式的规则将两者相加，可得 D+N=A，这里的 A 才是有效地址，最后再根据 A 访问主存读取操作数 S。

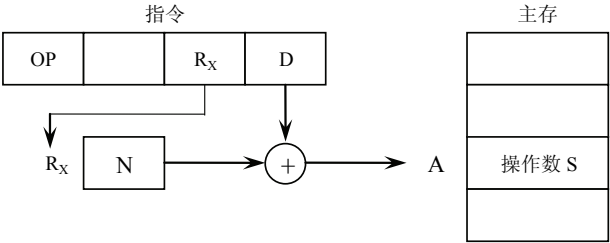
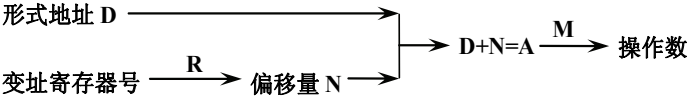


图 3-12 变址寻址方式示意图

变址寻址方式的寻址过程可以描述为：



【例 3-8】 下面分别列出了寄存器内容，以及主存储器数据区的地址和单元内容。若指令中给出的寄存器号为 000，形式地址为 1000H，按变址寻址方式读取操作数。

| | | | |
|--------------------|-------|------------|-------|
| 寄存器：R ₀ | 0030H | 主存单元：1000H | 7A00H |
| R ₁ | 1000H | 102FH | 1000H |
| R ₂ | 2000H | 1030H | 2C00H |

按照变址寻址方式的定义，指令指定的变址寄存器是 R₀，其中存放的位移量为 0030H，形式地址是 1000H。计算有效地址：D+N = 1000H + 0030H = 1030H，据此有效地址去访问主存储器，读得操作数 S = 2C00H。

变址寻址方式的应用很广，典型的用法是：将形式地址作为基准地址，如某个数组的首址，或者作为一个常量，而变址寄存器的内容作为偏移量（位移量、变址量），即目标单元与首址单元之间的距离。此时，形式地址段的位数应能提供全字长地址码，才可以覆盖完整的存储器编址空间；而变址寄存器提供的偏移量，其位数则可以少一些，只需覆盖目标数据区的存储空间即可。例如，某计算机主存容量为 2 GB，目标数据块大小为 512 B，可存放于主存的任一区域，则形式地址段应有 31 位，而偏移量只需 9 位。

但是，在定长的指令中，由于地址段位数有限，形式地址往往不能提供全字长的地址码。而实际计算机中变址寄存器的位数都能提供全字长地址码。因此，上述使用方式并非一成不变，可以根据实际需要灵活变化，关键的概念是：**有效地址 = 形式地址 + 变址量**。

【例 3-9】 对一个连续数组的操作，可以通过两条指令的简单循环得以实现。在第一条指令中，操作码指明操作类型，形式地址给出该数组的首址，指令还指定了谁充当变址寄存器并让偏移量的初值为 0。第二条指令用来修改变址寄存器的内容，使其递增或递减。如果沿地址增加的方向对数组进行访问，则第一次是对数组的第一个元素进行处理（其偏移量为 0），然后偏移量修改成 1；则第二次就是对数组第二个元素进行处理；以此类推，可将整个数组处理完毕。有的计算机还设置了自动变址方式，可将上述两条指令的功能组合起来，由一条指令即可完成，从而使编程实现更简洁。

【例 3-10】 查表操作。在企业管理系统中需要许多表格形式的信息组织，如一张按月汇总的销售金额表，打印出来可能是一张二维表格或者多维表格，但存放在主存储器中则是一组依次存放的元素。查表时可以采用变址寻址方式，形式地址为表在存储器中的首址，偏移量为目标元素所在单元与首址之间的距离，这个偏移量随月份的增加而增加，就可以从表中直接查找某月的销售金额，不需要遍历整个表存储区域。只需要修改变址寄存器内容，而指令本身不需做任何修改，就可以访问表格中任何一行所在的存储单元。

【例 3-11】 数据块搬迁。若要将一个数据块从一个存储区迁移到另一个存储区，可以通过两条指令（一读，一写）来实现，它们的形式地址段分别给出两个存储区的首址，两条指令使用同一个变址寄存器，通过修改偏移量，就能很方便地实现数据搬迁。

7. 基址寻址

基址寻址方式的定义是：指令中分别给出一个寄存器号和一个形式地址，寄存器中的内容作为基准地址，形式地址作为偏移量，将基准地址与偏移量相加作为操作数的有效地址，再按此地址访问主存，在相应的主存单元中读取或写入数据。

基址方式的指令格式如下：

| OP | R _B | D |
|-----|----------------|------|
| 操作码 | 基址寄存器号 | 形式地址 |

相应地，在这种寻址方式下，应在 CPU 中设置专用的基址寄存器，或者由程序指定某个通用寄存器担任基址寄存器。

【例 3-12】 下面分别列出了寄存器内容，以及主存储器数据区的地址与单元内容。若指令中给出寄存器号为 001，偏移量为 007FH，按基址寻址方式读取操作数。

| | | | |
|--------------------|-------|------------|-------|
| 寄存器：R ₀ | 1000H | 主存单元：2000H | AC00H |
| R ₁ | 2000H | ⋮ | ⋮ |
| R ₂ | 3000H | 207FH | 7A3CH |
| | | ⋮ | ⋮ |
| | | 3000H | B1C0H |

按基址寻址方式的定义，指令指定的基址寄存器是 R₁，它提供的基准地址为 2000H，指令中给出的偏移量为 007FH，两者相加得到有效地址 207FH，再根据这个有效地址访问主存单元，可得到操作数 7A3CH。

大家可能要问，从表面上看，基址寻址方式与变址寻址方式的有效地址计算法几乎是一样的，都是指定寄存器内容与形式地址相加，只不过对寄存器内容与形式地址的叫法不同而已，那么基

址寻址方式究竟与变址寻址方式有什么区别呢？为了回答这个问题，我们分析一下为什么采用基址寻址方式。

一种典型应用是程序的重定位。前面曾经谈到，用户一般采用高级语言编制程序，经过编译成为用指令序列表示的目标程序，目标程序由操作系统调入主存运行。用户在编程时，并不知道目标程序将被安排在主存的哪段区域。因此，用户编程时使用的是一种与实际主存地址无关的逻辑地址，将来运行时再自动转换为操作系统分配给它的主存地址（物理地址），即程序重定位。此外，在多道程序运行方式中也存在程序重定位问题，因为这些程序可能是由不同的用户独立编制的，编程时只能使用逻辑地址，将来由操作系统决定调哪段程序运行，并进行程序重定位。显然，采用基址寻址方式能够较好地解决这个问题，即在实现程序重定位时由操作系统给用户程序分配一个基准地址，并将它装入基址寄存器，在执行程序时就可以自动映射成实际的主存地址。

基址寻址方式的另一种典型应用是扩展有限字长指令的寻址空间。当采用有限字长指令时，分配给地址段的位数有限，难以给出全字长地址码，故不能覆盖大容量主存的任一区域。采用基址寻址，可由基址寄存器提供全字长地址码，再由指令给出一个位数较短的偏移量，两者配合足以覆盖主存的任何区间。具体方法是在运行时让基址寄存器装入某个主存区域的首地址，并在指令中给出相对于首地址的偏移量。例如，主存容量 16 MB，基址寄存器 24 位，它提供的 24 位地址码足以定位任意地址的主存单元；指令中地址段 16 位，其中的 2 位用来指定 4 个基址寄存器之一；14 位形式地址给出位移量，可以访问一个 16 KB 的连续存储区，通过变换基址寄存器或修改基址寄存器的内容，就可以重新定位另一段 16 KB 的存储区域，各存储区域甚至可以部分重叠。由于在某段时间内被运行的程序往往集中在一个有限区域（即局部性现象），基于程序运行的这一特性，基址寻址方式能够显著提高寻址效率。

虽然变址寻址与基址寻址在形成有效地址的方法上很相似，但实际并不相同，差别在于：变址寻址中形式地址给出的是基准地址，寄存器给出的是偏移量；基址寻址中形式地址给出的是偏移量，而寄存器给出的是基准地址。

两种寻址方式的具体应用也不同：变址寻址方式立足于面向用户，可用于访问字符串、数组、表格等成批数据（或其中的某些元素）；基址寻址方式立足于面向系统，可用来解决程序在实际主存中的重定位问题，以及在有限字长指令中扩大寻址空间等。

8. 基址加变址寻址

在一些计算机中还设置了一些复合型的寻址方式，如基址加变址寻址等。

基址加变址寻址方式的定义是：指令中给出一个基址寄存器号 R_B 、一个变址寄存器号 R_X 和一个形式地址 D ，基址寄存器的内容作为基准地址，变址寄存器的内容作为变址偏移量，形式地址则作为常规偏移量。

指令基本格式如下：

| OP | R_B | R_X | D |
|-----|--------|--------|------|
| 操作码 | 基址寄存器号 | 变址寄存器号 | 形式地址 |

则操作数的有效地址=基准地址+变址偏移量+常规偏移量，再根据得到的有效地址去访问主存，从对应主存单元中读取或写入操作数。

这样的寻址方式可以方便地处理二维数组或表格。

【例 3-13】 若某商场的销售金额汇总情况如表 3-1 所示，采用基址加变址的寻址方式查询某天的销售额。

表 3-1 商场销售金额统计示意表

| 日 月 \ 金额 (万元) | 1 | 2 | ... | 17 | ... | 30 | 31 |
|---------------------|-----|-----|-----|-----|-----|-----|-----|
| 1 | 100 | 100 | ... | 80 | ... | 60 | 80 |
| 2 | 50 | 60 | ... | 100 | ... | — | — |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 6 | 60 | 80 | ... | 90 | ... | 80 | — |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 12 | 100 | 100 | ... | 100 | ... | 90 | 100 |

为了说明基址加变址寻址的基本原理，在表示时忽略了表的格式信息，只按二维数组格式在主存中裸存本年度每天的销售金额数字。假定表格在存储区的首址为 1000H（该单元存储的是 1 月 1 日的销售金额），每个主存单元存放一天的销售金额，考虑到月大月小等因素，固定为每月数据分配 31 个连续的存储单元，且从 1 月到 12 月依次连续存储。因此可以按照这种事先约定的存放格式去识别日期，而不必存放对应的日期信息。

现在就可以采用基址加变址的寻址方式来定位任意日期的销售数据。具体过程是：先指定 R_0 为基址寄存器，用来存放全年数据在主存中的首地址 1000H； R_1 为变址寄存器，用来存放第几月（取值 0~11）；形式地址用来表示第几天（取值 0~30）。如果要编程访问 6 月 17 日的销售数据，则设置 R_1 的内容为 $(5 \times 31)_{10} = (155)_{10} = (10011011)_2 = 009BH$ ，形式地址设置为 $(16)_{10} = 0010H$ ，则目标数据的有效地址 = $1000H + 009BH + 0010H = 10ABH$ 。

9. 相对寻址

相对寻址的定义是：将程序计数器 PC 当前的内容作为基准地址，指令中给出的形式地址为偏移量（可正、可负），两者相加后形成操作数的有效地址。这种寻址方式是以 PC 当前的内容为基准进行偏移定位（往前或往后）的，所以称为相对寻址。此时，由于操作数的有效地址也是把一个寄存器（即 PC）的内容与一个形式地址相加得到的，因此与基址寻址方式的原理相同，故有的计算机干脆把相对寻址当作是基址寻址的一种特例，只不过指定的基址寄存器是程序计数器 PC，其助记符一般为 X(PC)。

在分析相对寻址方式时，一定要注意基准地址指的是 PC 寄存器当前的内容，它与当前指令的主存地址并不一致。这是因为 CPU 在从主存中取指令时，每读取一次主存，PC 的内容会自动加 1，以便在指令执行结束后，PC 刚好能指向下一条指令。换句话说，如果从主存中读取指令后，PC 的内容仍然维持不变，那么 PC 就会始终指向同一条指令，从而导致该程序的后续指令无法被继续执行。

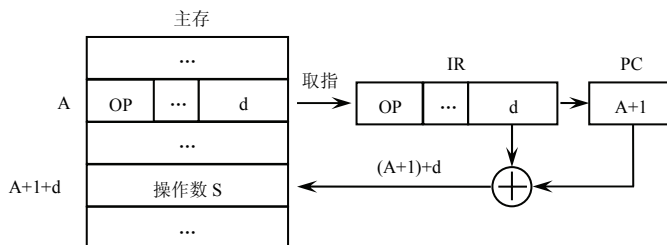


图 3-13 相对寻址方式示意图

如图 3-13 所示，若取指时 PC 的内容为 A，按地址 A 从主存中读取指令放入 IR，然后 PC 的

内容自动加 1。指令中的形式地址字段给了偏移量 d ，它表示从当前 PC 值（即 $A+1$ ）到操作数所在单元之间的距离。按相对寻址规则，操作数的有效地址= $A+1+d$ ，据此有效地址访问主存，即可从地址为 $A+1+d$ 的存储单元中读取到操作数 S 。

【例 3-14】 下面分别列出了寄存器内容，以及主存储器数据区的地址和单元内容。若从 1000H 单元中取出一条指令，该指令采用相对寻址方式读取操作数，形式地址 d （偏移量）为 0003H。

| | | | |
|--------|-------|------------|----------|
| 寄存器：PC | 1000H | 主存单元：OFFEH | BC00H |
| R_0 | 2000H | | \vdots |
| R_1 | 3000H | 1004H | AF00H |

按照相对寻址方式的定义，现行指令存放在 1000H 单元中，取指后 PC 的内容变为 1001H，因此基准地址是 1001H，指令中给出的偏移量为 0003H。

因此，操作数的有效地址= $1001H + 0003H = 1004H$ ，据此访问主存，得操作数 AF00H。

【例 3-15】 同上例的寄存器、主存单元内容，从 1000H 中读取指令，该指令采用相对寻址方式读取操作数，若形式地址（偏移量）为 -0003H。

操作数的有效地址 = $1001H - 0003H = OFFEH$ ，据此访问主存，得操作数 BC00H。

注意：如果该指令是一条多字节指令，占用多个主存单元，则取指时每读取一次主存，PC 的内容都要加 1，因此完成 4 字节指令的取指操作后，PC 的内容实际上已经在指令地址的基础上增加了 4。对于不同的计算，内存的编址方式或者指令的长度可能都互不相同，但对于相对寻址方式，应始终把握住基准地址是取指完成后 PC 当前的内容。

虽然我们是以寻找操作数为例来说明相对寻址方式的，实际上这种寻址方式在实现程序转移时使用最多。当我们编写到第 n 条程序指令时，如果需要返回到第 $n-d$ 条指令，或者跳到第 $n+d$ 条指令，就可以采用相对寻址方式来实现。

下面举例介绍两种常见的程序分支方法。

【例 3-16】 在程序的基本形态中有一种“循环”模式，即让一个程序段重复执行若干遍，直到终止条件满足时才退出循环。为此，常在该程序段的末端设置循环终止条件，如果条件尚未得到满足，就用相对寻址方式使程序返回到程序段的起始点重复执行，此时应设置偏移量为负值。如果循环的终止条件已满足，设置指令中的偏移量为正值，就能确保程序退出循环，继续执行循环体外的后续指令。

【例 3-17】 程序的分支常采取二路分支方式，一种方法是：当满足分支条件时 PC 内容加 1，这条后继指令是一条无条件转移指令，使程序转移到另一程序段；当分支条件不满足时，可用相对寻址方式使程序跳到第 PC+2 条指令处，再顺序往下执行。这样就实现了简单的两路程序分支。

10. 页面寻址

页面寻址方式的定义是：PC 的高位段作为高位段，指令给出的形式地址 d 作为低位段，操作数的有效地址由这两部分地址段拼接而成，其规则可描述为：

$$\text{操作数有效地址} = (PC)_H \cup d$$

【例 3-18】 下面分别列出了寄存器内容，以及主存储器数据区的地址与单元内容。若从主存单元中取出一条指令后，PC 的内容为 1030H，采用页面寻址方式读取操作数，指令中给出的形式地址为 FFH。

| | | | |
|--------|-------|------------|-------|
| 寄存器：PC | 1030H | 主存单元：10FFH | AC00H |
| R_0 | 2000H | 1100H | 7FC0H |
| R_1 | 3000H | | |

按照页面寻址方式的定义，取 PC 内容的高 8 位与形式地址相拼接，得到操作数有效地址 10FFH，再从主存中得到操作数 AC00H。

计算机中通常都采用了页式存储器管理技术，即将主存储器分为若干相同容量的页面，主存单元的地址就可映射成“页号+页内地址”。让 PC 内容的高位段对应主存的页号，访存时指令再给出目标单元的页内偏移量（页内地址，即该页起点到操作数所在单元之间的距离），采取页面寻址方式，就有利于快速生成有效地址以访问目标主存单元。

【例 3-19】 某机主存容量 1 MB，分为 1024 页，每页容量 1 KB。若采取页面寻址方式，则 PC 内容的高 10 位作为页面号，它也指明了当前程序运行所在的页面。指令中提供 10 位形式地址作为页内地址，两者相拼接就形成了长度为 20 位的有效地址。

11. 堆栈寻址

堆栈是一种按“后进先出”（从另一角度看就是“先进后出”）存取顺序进行存取的存储结构。一般的做法是：在主存储器中划出一段区间作为堆栈区，堆栈区有两端，作为起点的一端固定，称为栈底，在开辟堆栈区时由程序设定栈底地址；另一端称为栈顶，随着将数据压入堆栈，栈顶位置自底（地址码值较大）向上（地址码值减少）浮动；也就是说，堆栈区在主存储器中的位置可由程序设置，其大小可在一定范围内变化。对堆栈的读/写都是对栈顶单元进行的，读出又称为出栈，写入又称为压栈。为了指示栈顶的位置，在 CPU 中设置一个具有加、减计数功能的寄存器作为堆栈指针，命名为 SP（Stack Pointer），SP 中的内容就是栈顶单元地址。随着数据的压入或弹出，SP 中的内容将自动修改。按照前述自底向上的生成方式，若将数据压入堆栈，则 SP 内容减少，即栈顶向上浮动；若从堆栈中弹出数据，则 SP 内容增大，即栈顶向下浮动，弹出之后原栈顶单元内容视作不再存在。因此，对堆栈的压入就像是自底向上地堆放东西，相应地对堆栈的弹出像是从顶端取走东西；后存放的东西先取走，即后进先出（LIFO）；先存入的东西被压在栈底，即先入后出（FILO）。因此这种存储结构被称为堆栈。在对堆栈的基本概念简要介绍后，下面给出堆栈寻址方式的定义。

堆栈寻址方式是专门用于访问堆栈的寻址方式，操作数在堆栈中，指令隐含约定由堆栈指针 SP 寄存器提供栈顶单元地址，进行读出或写入。

如图 3-14 所示，如果现行指令要求从堆栈中读出一个操作数，并不需要给出堆栈地址，而是从 SP 中读取操作数地址 A，据此访问主存储器，从堆栈的栈顶单元读取操作数 S。读取后，SP 内容加 1，指向下一个单元即新的栈顶。

堆栈是一个重要的概念，这种存储结构广泛用于子程序的调用与返回、中断处理、逆波兰式计算等。因而几乎所有的计算机都在硬件上支持堆栈，有堆栈指针 SP。

随着向堆栈压入数据，堆栈的存储空间增大，称为堆栈生长。图 3-14 描述的生长方式属于“自底向上”生长方式，其特征是栈顶的地址码值小于栈底地址码值。习惯上，在画存储单元框图时，地址码值小的存储单元（如 0000H 单元）位于上面，地址码值大的存储单元（如 FFFFH 单元）位于下面，这与编写程序时的地址顺序相吻合。大部分计算机的堆栈都采取“自底向上”生长方式，但有的采取“自顶向下”生长方式或者“栈顶固定”生长方式。

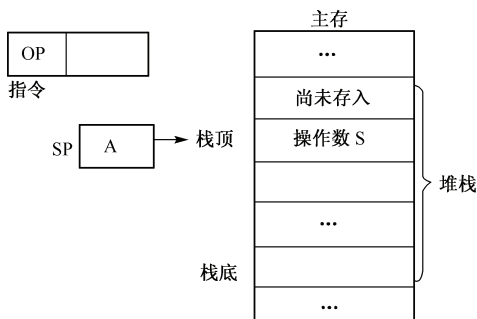


图 3-14 堆栈寻址方式示意图

下面通过一组例子进一步阐明堆栈的工作方式及其应用。

【例 3-20】 对堆栈的连续压入与连续弹出（自底向上生长方式）。

基本的堆栈操作指令有两种：压入指令 **PUSH**（进栈、压栈），将指定的操作数存入栈顶；弹出指令 **POP**（出栈），将栈顶数据读出，送入指定目的地。图 3-15 描述了这两种操作的典型过程。

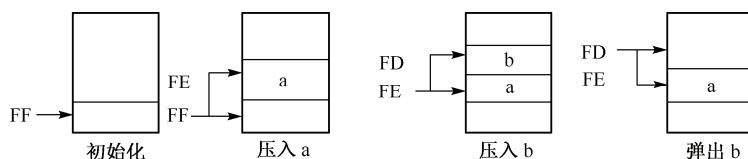


图 3-15 堆栈的压入与弹出

（1）初始化

将栈底地址即初始值送入堆栈指针 **SP** 寄存器，在本例中假定初始值为 **00FFH**。在某些实际系统中，将压入数据的第一个堆栈单元称为栈底，**SP** 则初始化为栈底地址+1。

（2）压入第一个数据元素 a

① **(SP)-1→SP**。即先修改堆栈指针，指向待存入的新栈顶。在本例中，**SP** 内容 **00FFH** 减 1 后，修改为 **00FEH**。

② 压栈。将 **SP** 的内容 **00FEH** 送入主存地址寄存器，将待存数据 **a** 送入 **00FEH** 单元。于是 **00FEH** 单元成为已存入信息的新栈顶。

（3）压入第二个数据元素 b

① **(SP)-1→SP**。**SP** 内容由 **00FEH** 修改为 **00FDH**。

② 压栈。将待存数据 **b** 送入 **00FDH** 单元，**00FDH** 单元成为新栈顶。

（4）弹出

① 将 **SP** 的内容 **00FDH** 送入主存地址寄存器，从栈顶单元将最后压入的数据 **b** 读出，送入指定地方。

② **(SP)+1→SP**。在弹出数据后再修改堆栈指针，即让 **SP** 内容加 1，由 **00FDH** 修改为 **00FEH**，指向存有数据的新栈顶。

例 3-20 所描述的堆栈工作过程体现了堆栈的存取顺序：后进先出（也就是先进后出）。因此，对这种堆栈的访问只能对栈顶进行操作，而不能直接访问堆栈中的任意单元，但随着堆栈的压入或弹出，栈顶位置是可以浮动的。

从例 3-20 中还可以看出，压栈的过程中先修改 **SP** 内容再写入数据。这就是前面说过的自减型寄存器间址方式 **-(R)**，先修改后操作。弹出的过程是先读出，再修改 **SP** 内容。这就是前面说过的自增型寄存器间址方式 **(R)+**，先操作后修改。只不过堆栈寻址方式中不需由指令给出 **SP** 寄存器号，它是由堆栈操作码（如压栈或弹出）隐含约定的，只要是对堆栈操作，就自动地由 **SP** 寄存器提供栈顶地址。至此也就回答了前面留下的一个问题：为什么自增型寄存器间址方式采取“先操作后修改”，而自减型寄存器间址方式采取“先修改后操作”，这正是为了适应对堆栈的弹出与压入。

【例 3-21】 软件堆栈。

通常计算机的指令系统中都有堆栈操作指令，相应地在 **CPU** 中设置一个堆栈指针 **SP** 寄存器，可隐含约定使用，于是我们认为：计算机的硬件直接支持堆栈操作，这样的堆栈常被称为硬件堆栈，用于程序调用与中断处理。但一个堆栈有时可能不够用，我们可以用自增型及自减型寄存器间址指令实现弹出及压入操作，从而用子程序软件建立新的堆栈区，这样的堆栈被称为软件堆栈。

具体的方法是指定某个寄存器（如 R_i ）担任软件堆栈的指针，向它赋予栈底初始值，当需要向这个堆栈压入新元素时，使用自减型寄存器间址方式 $-(R_i)$ 压入；当需要从这个堆栈中读出数据时，就使用自增型寄存器间址方式 $(R_i)+$ 弹出。这种软件堆栈可用于逆波兰式运算。

【例 3-22】 用零地址指令实现运算操作。

假定要实现一次乘法运算： $a \times b = c$ ，而操作数 a 与 b 依次存放在堆栈栈顶及其相邻单元（次栈顶）， $(SP) = 00F0H$ ，如图 3-16 所示，运算后这两个操作数均不再保留，栈顶用来存放乘积 c 。图 3-16 描述了这一操作过程。

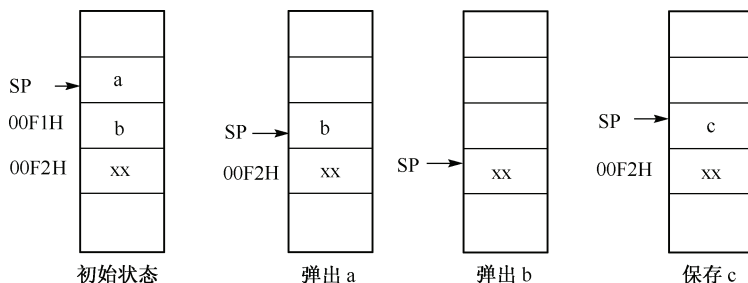


图 3-16 零地址指令运算操作示意图

- (1) 弹出第一个操作数 a ， SP 内容由 $00F0H$ 修改为 $00F1H$ 。
- (2) 弹出第二个操作数 b ， SP 内容由 $00F1H$ 修改为 $00F2H$ 。
- (3) 求乘积 c 。
- (4) 先修改指针， SP 内容由 $00F2H$ 修改为 $00F1H$ ，再将乘积 c 压入新栈顶 $00F1H$ 单元。

对堆栈的操作依靠隐含约定的 SP ，不需给出显地址，弹出的操作数和产生的乘积可以暂时存放在 CPU 的暂存器中，因此属于零地址指令。在逆波兰式计算中常采用这种方法。

【例 3-23】 堆栈用于子程序的多重调用。

在程序的基本形态中有一种是子程序调用，即将一些常用的、重复使用的程序段编写成可调用的子程序，它甚至可以为不同用户所共享。例如，一个求平方根的程序段，可能在一个用户程序中多处使用，为避免重复编写，就可以将它编成一个子程序。又如，多个用户都需要这一运算功能，可以事先编好，放在子程序库中作为系统的软件资源，供需要者调用，不必各自编写。调用者称为主程序。所谓调用子程序，即是转向该子程序入口，然后执行该子程序，在执行完时该子程序最后一条指令是一条返回指令，返回并继续执行主程序。因此，在转向子程序入口前必须把将来的返回地址保存在一个约定的地方，最好的方法就是将它压入堆栈中，因为堆栈的“先进后出”顺序特别适合多重调用的需要。

图 3-17 给出了一个子程序二重调用示意图，图上部显示了子程序调用时堆栈变化的情况，下部则显示了子程序返回时堆栈变化的情况。假设主程序在运行到 $1000H$ 处时需调用子程序 1，则将第一次转子程序的返回地址 $1001H$ 压入堆栈，假定是压入 $FFFFH$ 单元；若子程序 1 在运行到 $2010H$ 处时它自己又需要调用子程序 2，则第二重转子程序的返回地址 $2011H$ 压入堆栈，即存入 $EFFEH$ 单元。当子程序 2 执行完毕时，从堆栈中读取返回地址，即从 $EFFEH$ 单元中弹出返回地址 $2011H$ ，从而返回并继续执行子程序 1。当子程序 1 执行完毕时，从堆栈中再次读取返回地址，即从 $FFFFH$ 单元中弹出返回地址 $1001H$ ，从而返回并继续执行主程序。可见，堆栈的“后进先出”存取顺序非常适于多重嵌套的程序结构，有助于实现复杂程序组织。

【例 3-24】 向下生长方式的堆栈。

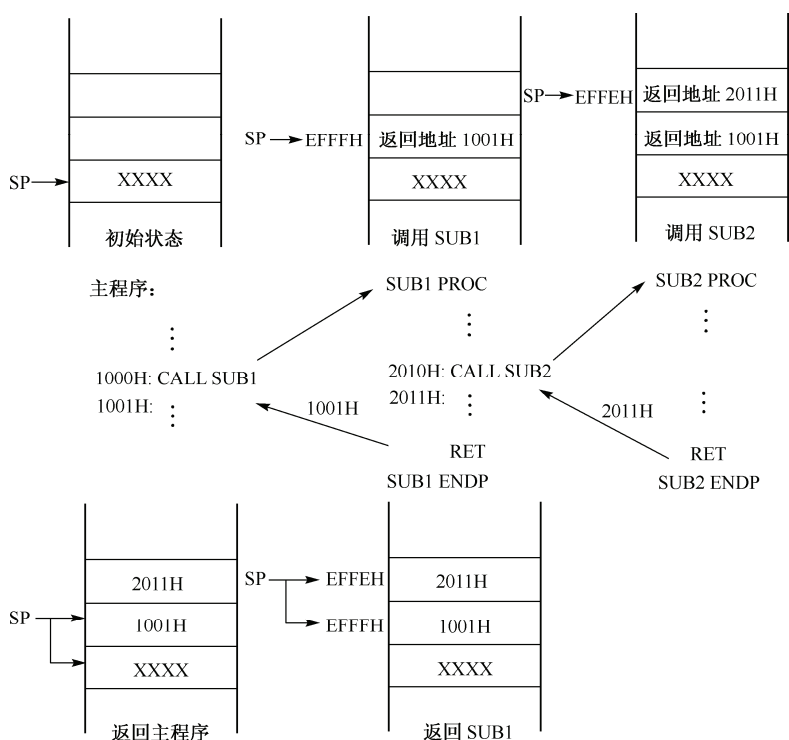


图 3-17 堆栈用于子程序的多重调用

如前所述，大多数计算机的堆栈属于“自底向上”生长方式，但有的计算机采用“向下”生长方式的堆栈。这种方式仍然是栈底固定、栈顶浮动，与“向上”生长方式不同的是：栈底在上而栈顶在下，即栈顶地址大于栈底地址，如图 3-18 所示。

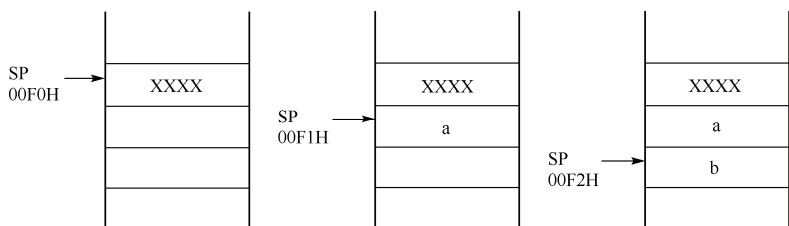


图 3-18 向下生长方式的堆栈

(1) 初始化

通过程序设置 SP 的初值，在本例中栈底为 00F0H。

(2) 第一次压入

- ① 令 $(SP)+1 \rightarrow SP$ ，使栈顶向下（地址增大）浮动。即 SP 内容由 00F0H 修改为 00F1H。
- ② 压入。向栈顶 00F1H 单元压入数据 a。

(3) 第二次压入

- ① $(SP)+1 \rightarrow SP$ ，栈顶由 00F1 修改为 00F2H。
- ② 压入。向栈顶 00F2H 单元压入数据 b。

(4) 弹出

- ① 从 SP 指示的栈顶 00F2 单元读出数据 b。
- ② $(SP)-1 \rightarrow SP$ ，即 SP 内容由 00F2H 修改为 00F1H。这样，00F1H 单元成为弹出后的新栈顶。

【例 3-25】 栈顶固定方式的堆栈。

在个别情况下，需要一种快速的小容量堆栈。例如，在第 3 章中将介绍微程序控制器，微程序中可能有多重微子程序的调用与返回，就需要这种堆栈。一般是用移位寄存器组来实现，如图 3-19(a)所示，有 $n+1$ 个寄存器， R_0 至 R_1 ， R_1 至 R_2 ， \cdots ， R_{n-1} 至 R_n 之间，各对应位存在进位关系，即可在同一时刻实现： R_0 内容移至 R_1 ，而 R_1 的原内容移至 R_2 、 \cdots 、 R_{n-1} 的原内容移至 R_n 。

图 3-19(b)将上述移位寄存器组画成堆栈框图，在这种堆栈中，栈底是浮动的，栈顶的位置是固定的，它是位于顶部的寄存器 R_0 。对堆栈的压入与弹出都是对栈顶(R_0)进行的。如图 3-19(b)所示，压入时，数据写入栈顶，栈中原有的数据依次向下推移一个单元。则新数据 b 写入栈顶，而栈顶原内容 a 移至下一单元。弹出时，从栈顶取走数据，而栈中原有的数据则依次向上推移一个单元，则从栈顶取出数据 b ，而下一单元的数据 a 则移至栈顶。这种堆栈的压入与弹出很像子弹的装弹与射击过程。由于栈顶位置固定，所以不需设置堆栈指针。显然，这种方式不适于用主存储器区间构成的堆栈，因为多个存储单元之间的依次推移要花费许多时间。

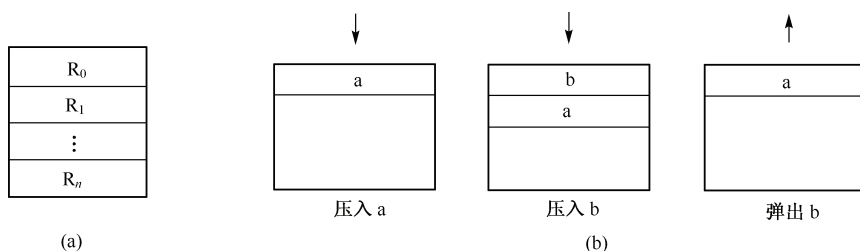


图 3-19 栈顶固定方式的堆栈

至此，我们介绍了多种常用的寻址方式，列举了大量实例，希望大家掌握它们的寻址过程，并能举一反三。

3.2.3 指令的功能和类型

一台计算机的指令系统中应该设置哪些指令？这体现了该计算机硬件所能实现的基本功能，也是编写程序的基本单位（包括由源程序转换得到的目标代码），因而指令系统体现了一台计算机的硬、软界面，也是不同机种 CPU 之间的主要差别所在。

从加强指令功能的角度出发，希望一个指令系统中包含尽可能多的指令，一条指令中含有尽可能多的操作命令信息。沿着这一思路发展，指令系统变得越来越复杂。具有这种复杂指令集合的计算机被称为 CISC (Complex Instruction Set Computer)。这种复杂性主要体现在以下两方面：

① 早期的指令系统只考虑对单个数据的运算操作，需要编制一个程序段或子程序才能处理某种数据结构。现在的一种发展趋势是：在指令中直接提供对各种常用数据结构的硬件支持能力，如对数组、表、队列、堆栈等的操作指令，从而向高级语言功能靠拢。有些计算机设置了专门的向量运算部件，指令中有相应的向量运算指令，可将某些适于向量化的计算任务，予以向量化并行处理。② 早期的指令系统主要面向用户的编程需要。现在的又一种发展趋势是：在指令设置方面增强对系统管理功能的硬件支持，如中断管理、操作系统的进程管理等。

从提高指令本身的执行效率角度出发，则希望指令比较简单，最好是能在一个时钟周期内执行一条指令。对程序实际运行情况所做的大量统计分析使人们惊奇地发现，机器所执行的指令竟然有 80% 以上是简单指令，而复杂指令只占很小比例。于是从 20 世纪 70 年代开始出现了另一种发展趋势，即选取比较简单而有效的指令构成指令系统，相应地发展了一整套的措施，形成了精

简指令系统计算机 RISC (Reduced Instruction Set Computer)。现在, RISC 技术已成为高速微处理器的主流, 它广泛用于各种高档工作站中, 大规模并行处理阵列计算机也大多由多个 RISC 微处理器构成。

传统的计算机, 包括不久前的主流微处理器如 Intel 80386、80486 等, 还是沿着 CISC 技术的道路发展的。随着 RISC 技术的发展, Intel 公司的主流微处理器采取了两相结合的方法。

不同机种, 对指令的分类方法可能不同, 归纳起来大致有以下 3 类。

① 按指令格式分类。例如, 传统的小型机 PDP-11 将指令格式分为双操作数指令、单操作数指令、程序转移指令等。

② 按操作数寻址方式分类。例如, 传统的大型机 IBM370 将指令分为 RR 型(寄存器—寄存器型)、RX 型(寄存器—变址存储器型)、RS 型(寄存器—存储器型)、SI 型(存储器—立即数型)和 SS 型(存储器—存储器型)。然后, 在每类指令中再按操作功能分为若干种指令。

③ 按指令功能分类。现在的大部分微处理器将指令分为: 传送指令、访存指令、I/O 指令、算术运算指令、逻辑运算指令、程序控制类指令、处理机控制类指令等。

采用前两种分类方法, 有利于 CPU 解释与执行指令。但对用户使用指令编程来说, 更关心指令的功能, 因而第三种分类方法现在用得更多。下面采取按功能分类的方法介绍一些常见的指令类型, 请特别注意其中一些指令的设置方法, 如传输指令、I/O 指令等。在举例时我们有时采用助记符描述指令。

1. 传输类指令

传输类指令是计算机中最基本的指令, 用于实现数据传输操作。从编程的角度, 传输类指令是使用得最多的指令。从掌握计算机工作机制的角度, 计算机硬件操作基本上都可以归结为信息的传送, 例如一条四则运算指令, 它的操作无非是将指定的操作数按规定的途径流经运算器, 从而形成相应的运算结果, 再送入目的地。因此, 掌握了数据信息的传输方法, 如寻址方式的实现、传送路径及其控制, 对掌握 CPU 的整机概念及其工作机制就易如反掌了。

许多计算机将传输类指令划分为三大类。① 传输指令, 实现 CPU 中各寄存器之间的数据传送, 如 $R_0 \rightarrow R_1$ 。这类指令可以高速实现。② 访存指令, 实现对存储器的读出或写入, 如将数据从主存储器调至 CPU 的寄存器之中, 以后就可以在 CPU 中进行运算处理。③ I/O 指令, 将有关 I/O 接口中寄存器的内容输入到主机 (CPU 的寄存器中或主存储器中) 或是将数据从主机输出到 I/O 接口中。

有的计算机将这三种合并为一个大类, 可称为通用型数据传送指令。根据寻址方式区分它是在 CPU 内传送还是访存, 如一条传输指令的源与目的地都采用寄存器寻址方式, 则说明它是一条寄存器之间的传输指令; 如果一个地址采用寄存器寻址, 另一个地址采用寄存器间址, 则是一条访存指令。根据地址范围, 可判别它是访存指令还是 I/O 指令。例如, 16 位地址可寻址的空间是 64 K, 如果将其中高端的 1 K 空间划为 I/O 接口寄存器的地址码空间, 则当地址码高 6 位为全 1 时, 表明它要访问 I/O 接口, 否则访问主存。

初学者需要注意, 当传输指令将数据从源地址传送到目的地后, 源地址中的内容一般仍保持不变, 因此我们说: 传输指令实际上是“复制”指令。所以存放在主存中的程序可以被重复执行, 数据可以被多次使用。但是, 从堆栈中读取数据后, 由于堆栈指针往下移动, 虽然源数据并未改变, 但该数据仍被视为不再存在。

【例 3-26】 执行一条传输指令“MOV R_1, R_0 ”, 其功能是将 R_0 的内容传输至 R_1 。执行前后

的寄存器内容如下：

| | | | | | |
|------|----------------|-------|------|----------------|-------|
| 执行前： | R ₀ | AC00H | 执行后： | R ₀ | AC00H |
| | R ₁ | XXXX | | R ₁ | AC00H |

在具体设置传输指令时，一般应当对三方面做出约定或说明。

① 传输范围，即指令允许数据在什么范围内传送。如前所述，操作数的来源与目的地可能是 CPU 寄存器、主存储器或 I/O 接口寄存器。

② 传输单位。一般来说，数据可以按字节、字、双字长、数组为单位进行传输，因此传输指令应该用某种方式指明数据传输的单位。例如，在通用型传输指令中，用 **MOVB** 表示按字节传输，用 **MOVW** 表示按字传输（即多个字节组成一个字）。又如，将寄存器分为每个寄存器 8 位（1 字节），几个寄存器组成一个字，用“**MOV AL, BL**”实现字节传输（L 表示低 8 位），用“**MOV AX, BX**”表示按字传输（X 表示组成一个字的一组寄存器），用“**MOV EAX, EBX**”表示双字传输。

③ 设置寻址方式。有的计算机为各种寻址方式分类编号，如 0 型、1 型、……、*n* 型等，在指令中设置专门的寻址方式编码字段，说明是何种寻址方式，如代码 000 表示 0 型寻址；有的根据操作码隐含约定是何种寻址方式。

2. 访存指令

虽然前面在通用型数据传送指令中已经包含了访问存储器指令，现在我们将它作为专门的一类，做些补充。许多机器都设置了专门的访存指令，用于主存储器与 CPU 寄存器之间的数据传送，分为读、写两类。

① 加载指令（读存储器）。从主存储器某个单元将数据读出，送入 CPU 的某个寄存器中，常被称为加载（**LOAD**）。加载指令又可分为按字节、按字等几条指令。

② 存储指令（写入存储器）。将数据写入某个主存储器单元，称为存储（**STORE**），它也可分为按字节，按字等。

③ 弹出（**POP**）。从堆栈栈顶弹出数据，可视为读存储器的一个特例。

④ 压栈（**PUSH**）。将数据压入堆栈栈顶，可视为存储指令的一个特例。

3. 输入/输出（I/O）指令

为了将 CPU、主存储器、各种外围设备连接成一个系统，建立系统级整机概念，需要特别关注 I/O 指令的设置和应用方式。I/O 指令实现主机和各外围设备之间的信息传送，输入和输出都以主机为参考点。由外部将信息送入主机，称为输入；由主机将信息送至外围设备，称为输出。主机方面的数据发送地/接收地，既可以是 CPU 中的寄存器，也可以是主存储器。外围设备方面则通过 I/O 接口与系统总线相连，从而实现与主机的信息传输，所以外围设备方面的数据发送地/接收地一般是 I/O 接口中的寄存器。相应地，主机对外围设备的访问一般是对有关接口寄存器的访问。

I/O 指令所传输的信息大致可分为三类：数据、命令、状态。从系统的角度，外围设备一般受 CPU 控制，如启动打印机或终止打印机工作，因此 CPU 需要通过输出指令向 I/O 接口送出命令信息，再由接口产生相应的命令发送给外围设备。一台计算机究竟连接哪些外围设备，其类型与数量要根据实际需要而定，相应地需要哪些命令也因设备的不同而不同。所以，CPU 往往采取一种通用的方法，即通过输出指令经数据总线向 I/O 接口的有关寄存器发送代码，并约定代码各位的含义，如最低位是启动位，为 1 时启动设备。这样的代码字称为命令字，接口中的相应寄存

器称为命令字寄存器或控制寄存器。

对外围设备的简单控制，只需由 CPU 发出命令即可。但是对于比较复杂的控制任务，CPU 需要根据实际的工作状态进行判别，做出相应的调整和决策。因此，往往在 I/O 接口中设置一个状态字寄存器，以 0、1 代码形式实时地记录外围设备与接口的有关状态信息，如“打印机正在打印（暂时不能接收打印信息）”、“打印机已经打印完现有内容（可以接受新的打印信息）”、“打印机出错”等。CPU 可以采用输入指令，将接口中的状态字代码经数据总线取回，送入 CPU 的某个寄存器，以供判别。

主机与外围设备之间经常需要进行数据传输，如由主机向打印机输出打印信息，从键盘向主机输入按键信息等。

在简要介绍了主机与外围设备之间的连接方式、输入/输出信息类型之后，我们还要讨论 I/O 设备的编址方法、I/O 指令的设置方法、I/O 指令的应用方式。

（1）I/O 设备的编址方法

要访问外围设备，首先要考虑对设备的编址方法。如前所述，主机是通过 I/O 接口访问外围设备的，因此对 I/O 设备的编址方法实际上就是对 I/O 接口中寄存器及相应部件的编址方法，可以分为两类。

① 外围设备单独编址。

早期的做法是为每台外围设备分配一个设备码，该设备的接口中设置有限的几个寄存器，如寄存器 A、B、C。在 I/O 指令中给出设备码，并指明是哪个寄存器。

现在普遍采用的方法是：为各 I/O 接口中的有关寄存器分配一种 I/O 端口地址，即编址到寄存器一级。各台设备有自己的接口，一个接口可以占有若干个 I/O 端口地址，各接口所占有的端口地址数目可以不同。系统软件对各端口地址进行分配。在常见的微型计算机中通过地址总线低 8 位（或低 16 位）提供 I/O 端口地址，最多可有 256 种（或 64K 种）编址，对一般的微机系统已经足够了。因此，只要送出某个端口地址，就能知道选中了哪一个接口中的哪一个寄存器，也就知道选中了哪台设备。

② 外围设备与主存储器统一编址。

有的计算机采取统一编址方式，即将 I/O 接口中的有关寄存器与主存储器的各单元统一编址，为它们分配统一的总线地址。也就是将寻址空间划分为两部分，大部分为主存，一小部分留给 I/O 接口寄存器（I/O 端口）。例如，某计算机的地址总线为 20 位，则其相应的编址空间为 1 M；可将其中地址码最大的（高地址端）4 K 个（FF000H~FFFFFH）地址分配给 I/O 端口使用，其余 1020 K 个（00000H~FEFFFH）地址分配给主存使用。当地址总线送出某个总线地址时，根据其地址码就可以判定是访问主存还是访问 I/O 接口中的寄存器。如上例，寻址空间 1 M，则存储器物理容量还是为 1 MB，但其中高地址端的 4 K 空间因其对应的总线地址编码已被 I/O 端口占用了，因此这部分存储单元已不再可用。

（2）I/O 指令的设置方法

通常有三类常见的 I/O 指令设置方法，一台计算机可以选取其中的一种或数种。

① 设置专用的 I/O 指令。

大多数计算机的指令系统中都设置了专门的 I/O 指令，以支持对外围设备（I/O 接口寄存器）单独编址。这类指令是 I/O 操作专用的且明确存在（非借用内存操作指令完成 I/O 操作），故又称为显式 I/O 指令。相应地，I/O 指令的操作码明确规定某种 I/O 操作，在地址部分分别给出 CPU 寄存器号及 I/O 端口地址。例如，输入指令“IN R₀, n”的操作含义是：将端口地址为 n 的

I/O 接口寄存器内容送到 CPU 内部的 R_0 寄存器中。

采用专用 I/O 指令启动外围设备的方法有两种：一种是由操作码给出启动命令，另一种是用输出指令，从 CPU 寄存器向 I/O 接口的控制寄存器送出命令字，其中包含启动位及其他命令。

② 采用通用的数据传输指令实现 I/O 操作。

有些计算机采用通用的数据传输指令实现 I/O 操作，相应地将外围设备（I/O 接口寄存器）与主存单元统一编址。如果传输指令的源地址是 CPU 寄存器，而目的地是接口寄存器，则这条传输指令就是一条输出指令。例如，传输指令“MOV n, R_0 ”，源地址为 CPU 中的 R_0 寄存器，目的地址 n 指向某接口寄存器。反之，如果传输指令的源地址是接口寄存器，目的地是 CPU 中的寄存器，则是一条输入指令，如“MOV R_0, n ”。

这类指令的 I/O 等价功能是借用内存传输指令实现的，所以把它们称为隐式 I/O 指令。

在通用传输指令中并不直接包含启动外围设备的启动命令，它们启动外围设备的办法是通过数据总线向接口送出命令字，其中包含启动位。

【例 3-27】 假设某外围设备接口中有 3 个寄存器，宽度均为 8 位，它们通过数据总线与 CPU 相连接，其总线地址的分配情况如下：

| | |
|--------|-------------|
| 数据寄存器 | 总线地址 FF000H |
| 命令字寄存器 | 总线地址 FF001H |
| 状态字寄存器 | 总线地址 FF002H |

执行“MOV FF001H, R_0 ”：将 R_0 的内容输出到地址为 FF001H 的接口命令字寄存器， R_0 中的内容是命令字（8 位），其中最低位 D_0 是启动位，为 1 则启动外围设备。

执行“MOV R_1 , FF002H”：将地址为 FF002H 的接口状态字寄存器的内容（8 位）输入到 CPU 内部的 R_1 寄存器中，供分析判断。

执行“MOV R_2 , FF000H”：将地址为 FF000H 的接口数据寄存器中的内容输入到 CPU 内部的 R_2 寄存器。

执行“MOV FF000H, R_3 ”：将 R_3 中的数据输出到地址为 FF000H 的接口数据寄存器，再传输给外围设备。

主机在与外设交互信息时，为了减轻 CPU 在 I/O 方面的工作负担，现代计算机中常设置一种专门用于管理 I/O 操作的协处理器，即 IOP（I/O Processor，输入/输出处理机），在大规模计算机系统中甚至设置专门的外围处理机。相应地，设置的专用 I/O 指令可以分为两级：一级是 CPU 管理 IOP 的 I/O 指令，负责启动及停止 IOP 等操作，这类指令的操作类型较少，功能比较简单；另一级是 IOP 执行的指令（如通道程序等），负责控制外围设备具体的 I/O 操作，这一级的指令功能相对丰富一些。

注意： CPU 的 I/O 指令（不管是显式还是隐式）都是通用的，它们一般不专门针对某一台具体的设备或某一种具体的操作。因为一台计算机系统中究竟连接哪些外围设备？多少台？有哪些具体操作？变化是很多的，而且是计算机系统设计者事先无法确定的。解决的办法就是用 I/O 指令向接口发送命令字，再转化为与具体设备相匹配的命令。对不同的设备，命令字的约定不同。

4. 算术逻辑运算指令

计算机的基本任务是对数据进行运算处理，计算机的运算分为算术运算、逻辑运算两大类，其中还包含了算术移位和逻辑移位。

（1）算术运算指令

几乎所有计算机都设置有这些最基本的算术运算指令：定点加（ADD）、减（SUB）、加 1（INC）、

减 1 (DEC)，求补 (NEG)、比较大小等。现在的主流微型计算机还设置了：定点乘、除，十进制运算，浮点加、减、乘、除等运算指令。巨型机中可能有向量运算指令，可以对整个向量或矩阵进行求和、求积等运算。

每次运算的单位可以是字节、字、双字等，对更高精度的多位字长运算则多数是通过软件子程序来实现的。

复杂的数学问题通过相应的算法可以分解、转换为一系列基本的算术运算，从而可用上述指令予以实现。较常使用的一些算法往往会由软件生产厂家事先编制成一些子程序（常称为例行子程序），作为软件开发平台的一部分纳入子程序库中，用户只需调用即可，不必自己编制。

早期的计算机为了降低成本，曾经采用两种方法。一种是“硬件软化”，即简化硬件，让 CPU 只设置一些基本的运算指令，复杂一些的运算功能如浮点运算则通过子程序实现。另一种办法是将计算机分成几档，最基本的 CPU 只执行一些基本的运算指令，较高档的 CPU 配备一些“扩展运算器”，CPU 有相应的扩展运算指令可调用扩展运算器，或将扩展运算器当成“外围设备”，通过 I/O 指令调用它们。

现代计算机采用超大规模集成电路 (VLSI) 技术，硬件成本大大下降，而获得高速运算能力成为首要问题。相应地，指令系统包含了更多的运算功能，改而采取“软件硬化”策略，但对更复杂一些的运算仍沿用扩展运算器的方法，现在称为“协处理器”。技术发展呈现这样一种趋势：随着 CPU 的功能不断升级，原来的一些协处理器功能（如浮点运算）被纳入 CPU 之中。

(2) 逻辑运算指令

虽然逻辑函数可以变化无穷，但都可以由与、或、非这三种最基本的逻辑函数予以组合实现。而异或逻辑则是一种应用很多的逻辑函数，可用来判别两字符代码是否符合、修改等。所以，计算机通常都设置 4 种最基本的逻辑运算指令：与 AND、或 OR、非 COM、异或 EOR。

上述指令都是按位进行逻辑运算的，各位之间没有进位、借位关系，因此又称为位操作指令类，有的计算机按位操作功能设置了：位测试、位分离、位清除、位设置、位修改等指令。

① 利用“与”运算实现按位测试。按位测试是指：测试某指定位是否为 1。例如，在分析状态字时，需要检测有关位是 1 还是 0，这可以通过“与”操作来实现。AND 指令有两个操作数，我们将被检测代码作为目的操作数；根据需检测的是哪一位，设置相应的屏蔽字，并将它作为源操作数。屏蔽字中对应于待检测位的屏蔽位为 1，其他位为 0，然后两个操作数相与。运算结果使需要检测的位保留原来的状态，不需检测的位由于被屏蔽所以为 0。如果运算后各位为全 0，说明被检测位为 0；否则，被检测位为 1。

【例 3-28】 位测试。

| | |
|---------|-------------------|
| 目的操作数 A | 1100 <u>1</u> 010 |
| 屏蔽字 B | 0000 <u>1</u> 000 |
| A AND B | 0000 <u>1</u> 000 |

② 利用“与”运算实现按位分离。有时需要从一个字中取出我们感兴趣的一段代码，称为位分离，这也可利用“与”运算来实现。让屏蔽字中对应于分离段的各位为 1，其余位为 0。

【例 3-29】 分离出低 4 位。

| | |
|---------|-------------------|
| 目的操作数 A | 1100 <u>1</u> 010 |
| 屏蔽字 B | 0000 <u>1111</u> |
| A AND B | 0000 <u>1</u> 010 |

③ 利用“与”运算实现位清除。位清除是指：将指定位清除为 0，也可利用“与”运算来实现。但屏蔽字的设置与位测试相反，被处理的数中哪些位需要清除，则屏蔽字中的相应位为 0，

其他位为 1。

【例 3-30】 位清除。

| | |
|---------|-------------------|
| 目的操作数 A | 1100 <u>1</u> 010 |
| 屏蔽字 B | 1111 <u>0</u> 111 |
| A AND B | 1100 <u>0</u> 010 |

④ 利用“或”运算实现位设置。位设置是指：将指定位设置为 1。可以让屏蔽字的相应位为 1，其余位为 0，然后与被修改代码相或。

【例 3-31】 位设置。

| | |
|---------|-------------------|
| 目的操作数 A | 11001 <u>0</u> 10 |
| 屏蔽字 B | 00000 <u>1</u> 00 |
| A OR B | 11001 <u>1</u> 10 |

⑤ 利用“异或”运算实现位修改。位修改是指：将指定位变反。例如在从存储器读取数据时，或在数据通信过程中，常需对数据进行校验，如果发现某一位出错，可通过将其变反而得到修正。这是因为二进制代码非 0 即 1，如果错了，变反就是修正。大家知道， $1 \oplus A = \bar{A}$ ，利用这一逻辑关系可实现变反。具体的方法是：被处理的数中哪些位需要变反，则屏蔽字中的相应位为 1，不修改的位为 0，然后两数相异或。这样，凡是与 1 异或的位都变反（由 0 变 1 或由 1 变 0），与 0 异或的位仍保持原来的状态不变，这就达到了修改的目的。

【例 3-32】 位修改。

| | |
|---------|-------------------|
| 目的操作数 A | 1100 <u>1</u> 010 |
| 屏蔽字 B | 0000 <u>1</u> 000 |
| A EOR B | 1100 <u>0</u> 010 |

⑥ 利用“异或”运算判符合。在程序中常需要识别字符、字符串，或是查询，这就需要待判定的字符代码与设定的字符代码进行比较，判定它们是否相同，称为判符合。异或运算正好能实现这一判定，如果异或的结果各位均为 0，则表明两组代码相同，否则为不同。

【例 3-33】 符合判断。

| | |
|---------|----------|
| A | 11001010 |
| B | 11001010 |
| A EOR B | 00000000 |

(3) 移位指令

移位也是一种基本操作，如在乘法中需要右移，在除法中需要左移，在代码处理中也可能需要移位操作。在一些计算机中将移位指令归入算术逻辑运算指令一类。移位又分为算术移位和逻辑移位。

① 算术移位指令。算术移位的对象是具有数值大小的数，因此在移位后会发生数值大小的变化。对于二进制数，每左移一位，数值由 x 增至 $2x$ ；每右移一位，数值由 x 减至 $x/2$ 。如果是带符号数，在移位过程中数符不变。

② 逻辑移位指令。逻辑移位使代码序列作循环移位或非循环移位，它只是使数码位置发生变化，没有正、负性质，也没有数值大小问题。换句话说，参与移位的代码序列被视为纯逻辑意义上的代码组合。例如，将 32 位数据通过移位以串行方式逐位输出，或者通过移位将串行输入的数据组装成可并行处理的 32 位数据。又如，通过循环移位使某些数位移至便于处理的位置等。

移位分为左移、右移。可以一次只移一位，也可以按指令中规定的位移量一次移若干位。

(4) 串操作指令

为了实现对数组元素的操作，许多计算机设置了串操作指令，加上重复前缀 REP，能对数组

进行传输、比较、扫描、装入、存储、输入、输出等串操作。组成数据串的元素称为串元素，它可以是字节、字、双字等。

【例 3-34】 在 8086/8088 中隐含约定：用寄存器 SI 作为源数据串的地址指针，寄存器 DI 作为目的地指针，串的长度存放在寄存器 CX 中。每执行一次串操作指令，SI 和 DI 的值便自动修改，指向下一个串元素单元。当在串操作指令之前加上重复前缀时，使用 CX 保存串长度，每执行一次串操作指令，CX 中的串长度值自动减 1，并且重复执行该串操作指令，直至 CX 的内容为 0。这样，串传送指令“REP MOVSB”可将整个数组从源存储区传送到目的存储区。

(5) 专用的数据处理指令

在一些主要用于数据处理的计算机中，还专门设置了下述指令。

① 转换指令：实现数制转换（如二进制与十进制之间的转换）或数据类型转换（如整型数与浮点数之间的转换，字节、字及双字间的转换等）。

② 检索指令：以给定的参考量作为依据，对一组信息进行检索。

③ 编辑指令：将一种格式的字符或数据编排为另一种格式的字符或数据，或者执行插入、删除、添加等编辑修改操作。

5. 程序控制类指令

前面讲的几类指令是用于对数据进行操作，程序控制类指令是用来控制指令的执行顺序，即选择程序的执行方向，并使程序具有测试、分析和判断的能力。比如，在什么情况下程序要进行转移，往何处转移等。

(1) 转移指令

在程序的执行过程中，通常采用转移指令来改变程序的执行顺序。它分为无条件转移和条件转移两类。

① 无条件转移指令。指令中给出转移命令（操作码）和转移地址，转移地址可以用多种寻址方式给出。程序执行到这条指令时就无条件地（强迫地）转移到指定的地址，即将该转移地址送入程序计数器 PC，再往下执行。

② 条件转移指令，主要用于程序分支。当程序执行到某处时，可能要在两条通路中选择一条，这就需要根据某些条件进行测试判断。典型的方法是：在条件转移指令中给出转移条件和转移地址；如果满足转移条件，则转向指令给出的转移地址；如果不满足转移条件，则程序继续按原来的顺序执行。

转移条件主要来源于 CPU 内部的一组特征触发器，又称为标志位，它们是 CPU 程序状态字 PSW 的基本组成，常见的有：进位触发器（为 0 表示没有进位或借位，为 1 表示有进位或借位）、溢出、结果为 0、结果为正或负等。在执行运算指令时，将运算结果的有关特征记入上述特征触发器，它们构成了一组可能用到的转移条件。相应地，常见的条件转移指令有：有进位转移 JC、无进位转移 JNC、有溢出转移 JO、无溢出转移 JNO、为零转移 JZ、不为零转移 JNZ、为正转移 JNS、为负转移 JS 等。

③ 循环指令。循环指令可以看作特殊的条件转移指令，指令中给出循环执行的次数，或者指定某个计数器作为循环次数控制的依据。例如计数器的初值为循环次数，每执行一次，计数器内容减 1，当计数器内容为 0 时停止循环。执行一条循环指令的操作包括：修改计数值、测试计数值，根据测试的结果控制继续循环或者退出循环。

(2) 转子程序指令与返回指令

例 3-22 中简要介绍了子程序的一些基本概念。为了调用某个子程序，需要执行一条转子程序

指令（简称转子指令）。而为了返回主程序，子程序最后需执行一条返回指令。

① 转子程序指令。从指令格式看，转子程序指令与无条件转移指令非常相似，指令中给出操作码以及转移地址，后者是子程序的入口地址。但从执行方式看，子程序执行完后要返回主程序，这点与无条件转移（不返回）是不同的。因此在转入子程序时应先把返回地址保存起来。以便当子程序执行完毕时，能用一条返回指令取出返回地址，使主程序从该地址继续执行。有几种保存返回地址的方法，目前广泛使用的办法是：将返回地址压入堆栈保存，因为堆栈“后进先出”的存取顺序非常适合子程序多重调用的需要。

② 返回指令。子程序的最后一条指令是返回指令，它只有操作码，因为返回地址是隐含获得的。如果转子指令将返回地址压入堆栈保存，则与之配套的返回指令就是从堆栈弹出返回地址，即将堆栈指针 SP 的内容作为地址，从栈顶读取返回地址。

（3）软中断（程序自中断）指令

后面将要讲到中断方式，这是一个非常重要的概念。引起中断的原因有多种，其中一种是由于程序执行一条软中断指令，所以又称为程序自中断。例如，软中断指令“INT n ”，除操作码 INT 外，指令还给出一个中断号 n ，根据它可以找到中断处理程序入口地址。执行软中断指令时，先将被中断的程序的断点压入堆栈，然后根据中断号 n 找到中断处理程序入口地址，送入 PC，转去执行中断处理程序。中断处理程序的最后一条是返回指令，它从堆栈中取出返回地址，即原程序的断点，然后继续执行原来的程序。

软中断的操作与转子程序很相似，不同之处是，这种软中断指令可以随机地插入到程序的任何位置。

早期，这种软中断指令主要用于程序调试中，如利用软中断指令为程序设置断点。编制了一段程序后，往往需要分段调试，可以在需要设置断点处临时插入一条软中断指令“INT n ”。当程序执行到该指令时，即按照响应中断的操作，暂停执行原来的程序，将其断点与有关寄存器内容（被称为“现场”）保存起来，如压入堆栈；然后转入相应的中断处理程序，执行一种调试跟踪程序，显示前一段程序的执行结果，分析并解决可能存在的错误。

现在，软中断指令还广泛用于系统功能调用。操作系统为用户提供了许多常用的系统功能，如磁盘调用、打开文件、复制文件等，可由用户在程序中以软中断指令“INT n ”调用，称为系统功能调用，编号 n 表明了不同的功能调用。

（4）控制处理机某些功能的指令

例如，对 CPU 状态字某些标志位的清除、设置、修改；空操作指令 NOP（除了消耗执行时间外，没有其他实质性操作）；实现 CPU 与外部事件的同步功能，如暂停 HLT、等待 WAIT、总线锁定 LOCK 等指令。

（5）面向操作系统的一些指令

计算机中的程序可分为系统程序与用户程序，前者如操作系统，是由系统程序员编写的，不能被用户程序所破坏。相应地，有些特权指令只能在操作系统中使用。

- ① 访问系统寄存器的指令，如访问系统控制寄存器、全局描述符表寄存器、任务寄存器等。
- ② 检查保护属性的指令，如检查某个数据段可否被读出、可否写入，调整段的特权级等。
- ③ 用于存储管理的指令。

3.3 CPU 的基本模型

前两节横向讨论了 CPU 组织的有关方面：既涉及其基本的逻辑组成，也涉及工作原理；既讨

论了内部组成,包括通用寄存器和数据通路结构等,也介绍了 CPU 与外部的连接及控制方式。本章的后面几节将通过一台模型机的设计,进一步建立整机概念,并具体深入讨论 CPU 的工作机制,即如何执行指令,如何为此产生微命令序列。

作为教学用模型机设计,我们将重点放在寄存器级,略去许多繁琐细节,并力求规整,采取较简单的组成模式,以尽量简洁的设计帮助读者掌握基本原理。与实际机器相比,模型机在每一时钟周期内所能完成的操作有限,所以指令执行速度较慢。读者在掌握基本原理之后,可以进一步考虑优化的途径,对有关逻辑细节也可自行补充完善。

3.3.1 CPU 设计步骤

设计一个 CPU 需要考虑机器的指令系统、总体结构、时序系统等诸多问题,最后达到形成控制逻辑的目的。因此,通常按下述步骤进行设计。

① 拟定指令系统。一台计算机的指令系统表明了这台机器所具有的硬件功能。例如,指令系统中包括乘、除运算指令,表明乘法、除法可以直接由硬件完成;如果指令系统中没有乘、除指令,则乘、除运算只能通过执行程序来实现。因此在设计 CPU 时,首先要明确机器硬件应具有哪些功能,根据这些功能设置相应指令,包括确定所采用的指令格式、所选择的寻址方式和所需要的指令类型。

② 确定总体结构。为了实现指令系统的功能,在 CPU 中需要设置哪些寄存器?设置多少寄存器?采用什么样的运算部件?如何为信息的传送提供通路?这些问题都是在确定 CPU 总体结构时需要解决的主要问题。

③ 安排时序。由于 CPU 的工作是分步进行的,而且需要严格定时控制,因此设置时序信号,以便在不同时间发出不同的微命令,控制完成不同的操作。组合逻辑控制方式和微程序控制方式在时序安排上有区别,前者多采用三级时序划分,后者往往采用两级时序。在做这一步设计时,我们将分两种情况进行讨论。

④ 拟定指令流程和微命令序列。这是设计中最关键的步骤,因为需要根据本步的设计结果形成最后的控制逻辑。拟定指令流程是将指令执行过程中的每一步传输操作(寄存器之间的信息传输)用流程图的形式描述出来,拟定微命令序列是用操作时间表列出每一步操作所需的微命令及其产生条件。这些工作深入到了 CPU 的控制机制,是学习计算机组成原理课程需要重点了解和掌握的内容。

⑤ 形成控制逻辑。这是设计的最后一步,视组合逻辑控制方式或微程序控制方式而采用不同的设计方法。在组合逻辑控制方式中,将产生微命令的条件进行综合、化简,形成逻辑式,从而构成控制器的核心逻辑电路。在微程序控制方式中,则根据微命令来编写微指令,组成微程序,从而构成以控制存储器为核心的控制逻辑。

设计的前两步与产生微命令的控制方式无关,我们将其作为总体设计放在本节讨论。设计的前三步与采用组合逻辑控制方式还是微程序控制方式有关,随后分别进行讨论。

3.3.2 模型机的指令系统

1. 指令格式

从简单、规整出发,模型机采用定长指令格式,每条指令 16 位长,占据一个存储单元。由于指令字长有限,采用寄存器型寻址,即指令格式中给出寄存器号,根据不同的寻址方式形成相

应的有效地址。

模型机指令格式分为以下 3 个基本的大类，如图 3-20 所示。

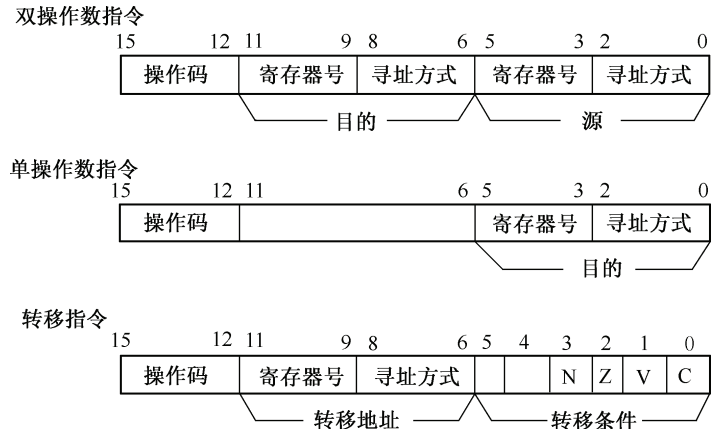


图 3-20 模型机三类指令的基本格式

① 双操作数指令格式：第 15~12 位是操作码字段，4 位操作码可以表示 16 种操作。第 11~6 位是目的地址字段，又可以分为两部分：其中 3 位用来给出寄存器的编号，其余 3 位用来表明寻址方式。第 5~0 位为源地址字段，与目的字段一样，也由寄存器编号和寻址方式这两部分构成。在图示的双操作数指令格式中，源字段和目的字段的位置安排与当前主流微机的做法相类似。

② 单操作数指令格式：第 5~0 位为目的地址字段，提供操作数和结果的存放地址，第 11~6 位未用（也可供扩展操作码用）。

③ 转移指令格式：第 15~12 位为操作码字段，第 11~6 位是转移地址字段（也包含寄存器号与寻址方式两部分），第 5~0 位为转移条件字段。其中第 3~0 位中某一位为 1，表明转移条件设置为程序运行的某种状态：进位 C，或溢出 V，或结果为零 Z，或结果为负 N。第 5 位指明转移方式，若为 0，表示相关标志位为 0 转移；若为 1，表示相关标志位为 1 转移。若第 5~0 位为全 0，表示无条件转移。后面讨论转移指令时再具体说明转移条件的设置。

2. 寻址方式

模型机寻址方式的特点是在指令中直接给出寄存器编号，供 CPU 编程访问。可编程寄存器包括通用寄存器 R₀~R₃、堆栈指针 SP、程序计数器 PC、程序状态字 PSW。针对同一种寻址方式编码，指定不同的寄存器，可以派生出多种不同的寻址方式。我们将模型机常用的寻址方式汇集于表 3-2 中。

(1) 0 型 (000)：寄存器寻址

操作数存放在指定的寄存器中。这种寻址方式可用来设置初始值，如设置某个寄存器的内容，或设置堆栈指针，或设置程序起始地址，或设置程序状态字等。

(2) 1 型 (001)：寄存器间址

操作数地址存放在指定的寄存器中，操作数则放在由该地址所指示的存储单元中。因此这种寻址方式需要按寄存器内容访存，从主存单元读取操作数，或将数据写入主存单元。

(3) 2 型 (010)：自减型寄存器间址

将指定寄存器的内容减 1 后作为操作数地址，再按此地址访存，从主存中读取操作数，或将数据写入主存。在汇编符号中减号在括号之前，形象地表示先减后访存。

表 3-2 模型机常用寻址方式汇总

| 类型编号 | 寻址方式 | 助记符 | 可指定的寄存器 | 定 义 |
|-----------|-------------|--------|---------------------------|-------------------------------|
| 0 型 (000) | 寄存器寻址 | R | $R_0 \sim R_3$ 、SP、PC、PSW | 寄存器的内容为操作数 |
| 1 型 (001) | 寄存器间址 | (R) | $R_0 \sim R_3$ | 寄存器的内容为有效地址 |
| 2 型 (010) | 自减型寄存器间址 | -(R) | $R_0 \sim R_3$ | 寄存器的内容减 1 后作为有效地址 |
| | | -(SP) | SP | SP 的内容减 1 后作为堆栈的栈顶地址 |
| 3 型 (011) | 立即/自增型寄存器间址 | (R)+ | $R_0 \sim R_3$ | 寄存器的内容为有效地址，访问该地址单元后寄存器的内容加 1 |
| | | (SP)+ | SP | SP 的内容为栈顶地址，出栈后 SP 内容加 1 |
| | | (PC)+ | PC | PC 的内容为立即数地址，取数后 PC 内容加 1 |
| 4 型 (100) | 直接/自增型双重间址 | @(R)+ | $R_0 \sim R_3$ | 寄存器的内容为间接地址，访问该地址后寄存器内容加 1 |
| | | @(PC)+ | PC | PC 的内容为间接地址，访问后 PC 内容加 1 |
| 5 型 (101) | 变址/相对寻址 | X(R) | $R_0 \sim R_3$ | 变址寄存器的内容与形式地址之和为操作数地址 |
| | | X(PC) | PC | 当前 PC 的内容与偏移量之和为有效地址 |
| 6 型 (110) | 跳步 | SKP | 无 | 执行再下一条指令 |

若指定的寄存器是 SP，则适用于压栈操作。将 SP 内容减 1 作为新栈顶地址，即可向新栈顶压入数据。若指定的寄存器是 $R_0 \sim R_3$ ，则可将指定寄存器当作反向指针使用；或将它当成堆栈指针，临时软件建栈用。

(4) 3 型 (011)：立即/自增型寄存器间址

操作数地址在指定寄存器中，访存后将寄存器内容加 1，作为新的地址指针。在汇编符号中，加号在括号之后，形象地表示先访存后加。这里采用了合并优化技巧，即利用指定寄存器的不同，派生出了立即寻址和自增型寄存器间址寻址两种方式。

若指定的寄存器是 PC，为立即寻址，操作数紧跟着指令。编程时，将操作数存放在紧跟指令的单元中，取指后 PC 内容加 1，则修改后的 PC 内容即为操作数的有效地址。根据该地址访存，读取操作数后，PC 内容再加 1，指向后继指令的存储单元。

若指定的寄存器是 SP，这种自增型寄存器间址方式可以作为堆栈弹出操作的寻址方式。从 SP 所指示的栈顶取出数据后，栈顶下浮，SP 内容加 1，指向新栈顶。

若指定的寄存器是 $R_0 \sim R_3$ ，则可将指定寄存器当作正向指针使用，或将它当成堆栈指针，用作软件的临时建栈。

(5) 4 型 (100)：直接/自增型双重间址

其中，汇编符号@是存储器间址的一种习惯标注符号。自增型双重间址是将指定寄存器的内容作为操作数的间接地址（即间址单元地址），根据该地址访存后寄存器内容加 1，指向下一个间址单元。双重间址需两次访存，第一次访存从间址单元中读取操作数地址；第二次访存再从操作数地址单元中取得操作数，或向该单元写入数据。

这里也采用了合并优化技巧，若指定的寄存器是 PC，就是直接寻址，操作数地址紧跟着指令。编程时将操作数地址存放在紧跟指令的单元中，取指后 PC+1。将修改后的 PC 内容作为地址，访问紧跟现行指令的存储单元（间址单元），从中取得操作数地址（称为绝对地址），据此再度访存，读取或写入操作数。然后 PC+1，指向后继指令。

若指定的寄存器是 $R_0 \sim R_3$ ，就是自增型双重间址方式，可将 R_i 作为查表的地址指针。

(6) 5 型 (101): 变址/相对寻址方式

其中, 汇编符号 X 是变址的一种习惯标注符号。在变址方式中, 形式地址存放在紧跟指令的存储单元中, 所指定的变址寄存器内容作为变址量, 将形式地址与变址量相加, 其结果为操作数的有效地址。再根据该地址访存, 读取或写入操作数。

这里也采用了合并优化。若指定的寄存器是程序计数器 PC, 就是相对寻址。取指后 PC+1, 以修改后的 PC 内容为基准地址, 从紧跟现行指令的存储单元中读取偏移量, 二者相加, 获得有效地址 (后继指令地址或操作数地址)。

由于有效地址是以 PC 值为基准地址再加上偏移量形成的, 若将程序段存放在另一个存储区中, 地址的相对关系不变。

若指定的寄存器是 $R_0 \sim R_3$, 就是常规的变址寻址方式。

(7) 6 型 (110): 跳步方式

现行指令执行后, 不是顺序执行下条指令, 而是执行再下条指令。因此在取指后 PC+1, 然后进行一次 PC+1, 使 PC 内容指向现行指令之后第 2 个单元。这是一种实现程序分支的方法。

3. 指令类型

根据模型机指令格式, 操作码有 4 位, 现用 14 种操作码表示了 15 种指令 (其中两种指令共用一个操作码), 余下 2 种操作码组合可供扩展。按操作数的多少, 模型机的指令可分为双操作数指令和单操作数指令两大类; 按指令本身的功能, 又可分为传输、运算、转移三类。各条指令的编码、助记符和指令含义见表 3-3。

表 3-3 模型机指令类型

| 操作码 | 助记符 | 含义 | 操作码 | 助记符 | 含义 |
|------|-----|----|------|-----|-----|
| 0000 | MOV | 传送 | 1000 | INC | 加 1 |
| 0001 | ADD | 加 | 1001 | DEC | 减 1 |
| 0010 | SUB | 减 | 1010 | SL | 左移 |
| 0011 | AND | 与 | 1011 | SR | 右移 |
| 0100 | OR | 或 | 1100 | JMP | 转移 |
| 0101 | EOR | 异或 | 1100 | RST | 返回 |
| 0110 | COM | 求反 | 1101 | JSR | 转子 |
| 0111 | NEG | 求补 | | | |

(1) 传输指令

由于可选用多种寻址方式, MOV 指令可用来预置寄存器或存储单元内容, 实现寄存器之间 (R-R)、寄存器与主存之间 (R-M)、各主存单元之间 (M-M) 的信息传输, 还可实现堆栈操作如 PUSH 和 POP, 不设专用的访存指令。在系统结构上将外围接口寄存器与主存单元统一编址, 因而 MOV 指令可用来进行 I/O 操作, 不再专门设置显式 I/O 指令。

(2) 双操作数运算逻辑指令

从 ADD 到 EOR, 共 5 条。ADD (加) 和 SUB (减) 都是带进位的。逻辑运算指令可用来实现位检测、位清除、位设置、位修正等位操作功能, 所用屏蔽字可由立即寻址方式提供。异或指令可实现两个逻辑量的符合判断操作。

(3) 单操作数运算逻辑指令

从 COM 到 SR, 共 6 条。

(4) 程序控制类指令

程序控制类指令包括转移指令、返回指令和转子指令, 用来实现程序的转移。

① 转移指令 JMP——用来实现无条件转移和条件转移。

在 JMP 的 $IR_3 \sim IR_0$ 中选择一位为 1，表明以 PSW 中的某一特征位作为转移条件。因此，JMP 指令第 3~0 位的含义与 PSW 第 3~0 位含义分别相对应，如 PSW 第 0 位是进位位 C，当 JMP 指令第 0 位（对应 C）为 1 时，表明以进位状态为转移条件。JMP 指令第 5 位（ IR_5 ）则决定转移条件是为 0 转（当 $IR_5=0$ ）还是为 1 转（当 $IR_5=1$ ）。例如，当 $IR_5=0$ 时表明转移条件为 $C=0$ （无进位）时转移；当 $IR_5=1$ 时则表明转移条件为 $C=1$ （有进位）时转移。若 JMP 指令第 5~0 位为全 0，则表示无条件转移。JUM 指令的转移条件设置情况如表 3-4 所示。

表 3-4 转移条件的设置

| IR_5 | IR_3 | IR_2 | IR_1 | IR_0 | 转移条件 |
|--------|--------|--------|--------|--------|-------------------|
| 0 | 0 | 0 | 0 | 0 | 无条件转移 |
| 0 | 0 | 0 | 0 | 1 | 无进位（ $C=0$ ）转移 |
| 1 | 0 | 0 | 0 | 1 | 有进位（ $C=1$ ）转移 |
| 0 | 0 | 0 | 1 | 0 | 无溢出（ $V=0$ ）转移 |
| 1 | 0 | 0 | 1 | 0 | 有溢出（ $V=1$ ）转移 |
| 0 | 0 | 1 | 0 | 0 | 结果不为 0（ $Z=0$ ）转移 |
| 1 | 0 | 1 | 0 | 0 | 结果为 0（ $Z=1$ ）转移 |
| 0 | 1 | 0 | 0 | 0 | 结果为正（ $N=0$ ）转移 |
| 1 | 1 | 0 | 0 | 0 | 结果为负（ $N=1$ ）转移 |

② 返回指令 RST——与 JMP 指令的操作码相同，实际就是 JMP 指令的特例。RST 指令只能采用自增型寄存器间址作为转移地址，并指定寄存器为 SP，即寻址方式为 $(SP)+$ ，则从堆栈中取出返回地址，然后 $SP+1$ 。

③ 转子指令 JSR——执行 JSR 指令时，先将返回地址压栈保存，然后按寻址方式找到转移地址（即子程序的入口地址），将它送入 PC 中（原则上也可对转子指令设置转子条件，与 JMP 指令相同）。

3.3.3 模型机的组成与数据通路

总体结构设计的内容包含确定各种部件设置以及它们之间的数据通路结构。在此基础上，就可拟出各种信息传送路径，以及为实现这些传送所需的微命令。

图 3-21 从寄存器级描述了模型机的数据通路结构（虚线框内部分），并标注了部分主要的微命令，寄存器部件的有关细节略去未画。这种数据通路结构采用了图 3-2 所示的结构，主要包括寄存器组、运算部件和内部总线等。我们有意选择一种简单而规整的 CPU 结构，以求掌握其工作机理与基本的设计方法，建立起 CPU 级的整机概念。

1. 部件设置

（1）寄存器

为简单起见，所有寄存器都是 16 位，内部结构是 16 个 D 触发器，代码输入至 D 端，CP 端同步打入。PSW 的特征位还可由 R、S 端置入，系统总线对 MDR、IR 的输入也可以分别由 R 和 S 端置入。

① 可编程寄存器——这些寄存器有 3 位编号，可以被 CPU 编程访问，包括：通用寄存器 R_0 （000）、 R_1 （001）、 R_2 （010）、 R_3 （011），堆栈指针 SP（100），程序状态字寄存器 PSW（101），程序计数器 PC（111）。

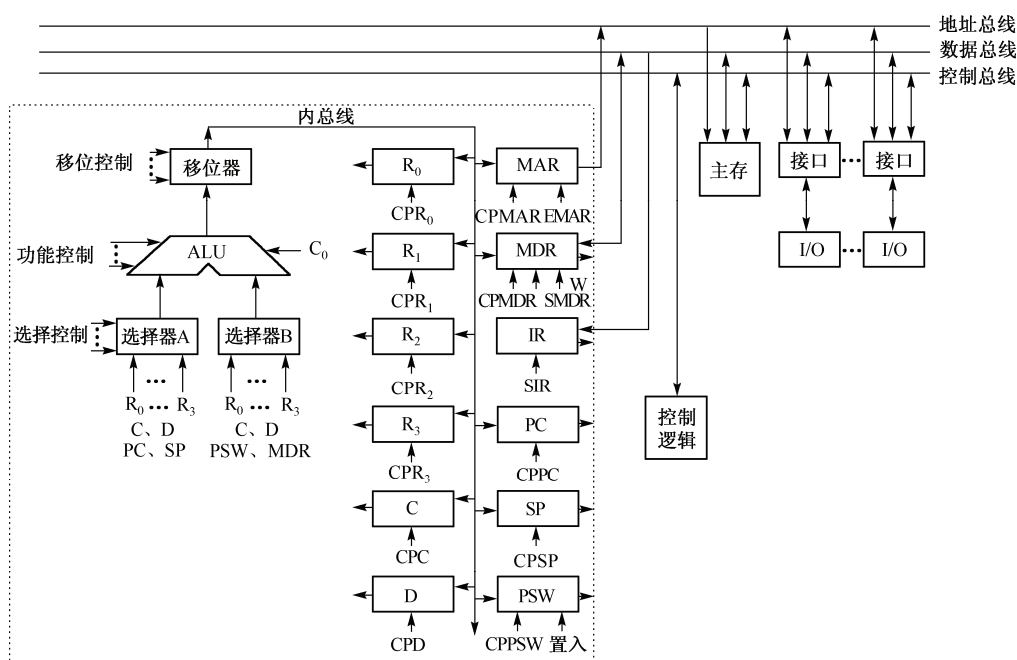


图 3-21 模型机数据通路结构

通用寄存器可提供操作数、存放结果、作地址指针、作变址寄存器等。

PSW 目前只用了 5 位，分别是：进位位 C（第 0 位）、溢出位 V（第 1 位）、结果为 0 位 Z（第 2 位）、结果为负位 N（第 3 位）、允许中断位 I（第 4 位，该位为 1 则开中断，为 0 则关中断）。其他位未用，留作扩展。

② 暂存器 C、D——模型机的特别安排，从主存中读取源操作数地址或源操作数时，就使用暂存器 C；从主存中读取目的操作数地址或者目的操作数时，以及需要暂存目的地址或运算结果时，就使用暂存器 D。

③ 指令寄存器 IR——为了提高读取指令的速度，将指令从主存中读出以后，经数据总线直接将其置入 IR。

④ 与主存的接口寄存器 MAR、MDR——CPU 访问主存的地址由地址寄存器 MAR 提供，MAR 连接地址总线的输出门是三态门。当微命令 EMAR 为高电平时，MAR 输出送往地址总线；当 EMAR 为低电平时，MAR 输出呈高阻态，与地址总线断开。

数据寄存器 MDR 既可以与 CPU 内的部件交换数据，也可以与系统总线交换数据。它一方面接收来自 CPU 内总线的代码（同步打入），或将代码送入 ALU 的 B 输入门；另一方面与数据总线间双向传送数据，既可在某些时钟周期内将 CPU 输出代码送往数据总线，也可在另一时钟周期内接收来自数据总线的代码（置入）。其输出级也采用三态门，可与数据总线断开。控制命令与操作的关系如表 3-5 所示。

表 3-5 MDR 的控制命令与操作的关系

| R | W | CPMDR | 操 作 |
|---|---|-------|--------------|
| X | X | 上升沿 | 将内总线数据送入 MDR |
| 0 | 0 | 0 | MDR 输出为高阻 |
| 0 | 1 | 0 | 向数据总线输出数据 |
| 1 | 0 | 0 | 数据总线数据置入 MDR |

表 3-5 中, R、W 分别是存储器读、写命令, CPMDR 是打入 MDR 的同步命令。不管有无读写命令, 只要出现 CPMDR 的上升沿, 便将内总线上的数据打入 MDR。如果 W 为 0, 则 MDR 的输出与数据总线断开; 若 W 为 1, 则 MDR 的输出与数据总线连接上, MDR 通过数据总线向主存写入数据。如果 R 为 0, 则 MDR 不从数据总线接收数据; 若 R 为 1, 则将总线上的数据置入 MDR, 因此 R 也可作为 MDR 的置入命令 (同 SMDR)。

选取由 R、S 端置入 MDR 的方式, 与置入 IR 一样, 是为了提高速度, 一旦从主存中读出数据有效, 就可立即置入 MDR。还有一个目的是出自教学考虑, 让输入方式多样化, 即除了常用的同步输入方式外, 还有 R、S 端异步置入方式。

(2) 运算部件

运算部件以算术逻辑运算部件 (ALU) 为核心, 还包括输入逻辑和输出逻辑。

① ALU 部件。模型机 ALU 采用 74181 型结构, 由微命令 M、S₀、S₁、S₂、S₃ 选择 ALU 操作功能, 采用负逻辑 (反变量) 输入, 可根据 74181 功能表找到要实现的功能与微命令之间的对应关系。C₀ 是送入最末位的进位信号, 可用来实现加 1 操作。

② 输入逻辑。ALU 输入端设置 A、B 两个多路选择器, 它们都具有八选一功能, 选择数据来源。有关寄存器的输出分别送往多路选择器的输入端, 以便送入 ALU 进行运算处理, 或通过 ALU 进行传输。通用寄存器 R₀~R₃ 和暂存器 C、D 既送往 A 选择器也送往 B 选择器, 处理起来比较方便。其他寄存器则只送往一个选择器。控制器将根据指令需要发出选择控制电位, 决定哪一个或哪两个操作数进入 ALU。

③ 输出逻辑。ALU 输出端设置一个移位器, 其逻辑结构也是一个多路选择器, 利用对应位的连接关系实现直传、左移 (左斜一位传送)、右移 (右斜一位传送)。

我们将在 3.4 节对运算部件所涉及的问题进一步讨论。

2. 总线与数据通路结构

模型机的数据通路采用总线结构。在 CPU 内部用内总线连接寄存器和运算部件, CPU 通过系统总线连接主存和外部设备。

(1) 内总线

模型机内部数据通路的特点是: 由 ALU 汇集各数据, 单向内总线实现数据分配, 寄存器在逻辑上分立。各寄存器将其输出分别送至 ALU 的输入选择器, ALU 输出经移位器送到内总线上, 内总线是 16 根单向数据传输线, 它们连接到各寄存器对应位的 D 输入端。数据究竟送入哪一个或哪几个寄存器, 取决于寄存器是否收到 CP 脉冲, 控制器只向需要接收数据的寄存器发同步输入脉冲。这种通路结构的优点是简单、规整、控制集中, 便于设置微命令; 缺点是只有一组基本数据通路, 并行程度较低。

(2) 系统总线

CPU 通过系统总线与外部连接, 如连接主存、各种外围设备。为简化模型机系统, 让 CPU 直接与系统总线相连, 不考虑信号的转换和扩展 (在实际系统中常需要)。

系统总线可分为地址总线、数据总线、控制总线三种。模型机系统总线采取同步控制方式。

CPU 通过 MAR 向地址总线提供地址, 以选择主存单元或外围设备 (接口寄存器)。外围设备 (如 DMA 控制器) 也可以向地址总线发送地址码。因此对 MAR 设置了控制命令 EMAR, 如前所述。

CPU 通过 MDR 向数据总线发送或接收数据, 由控制命令 R、W 决定传输方向及 MDR 与数据总线的断开和连接。主存 M 和外围设备也与数据总线相连, 向数据总线发送或接收数据。

CPU 及外围设备向控制总线发出有关控制信号,或接收控制信号。主存一般只接收控制命令,但也可提供回答信号。

3. 各类信息的传输路径

在确定了数据通路结构之后,可归纳出各类信息的传输路径。指令的执行基本上可以归结为信息的传输,即控制流和数据流两大信息流。控制流表现为指令信息的传输,以及由此产生的微命令序列。指令信息与数据信息的读取又依赖于地址信息。因此,清楚各类信息的传输路径对初学者很重要,它有助于从逻辑结构的角度了解指令是如何执行的,以及需要为此发出哪些微命令。参照图 3-21,模型机的指令信息、地址信息和数据信息的传送路径如下。

(1) 指令信息的传输

指令由主存读出,送入指令寄存器 IR: $M \rightarrow \text{数据总线} \rightarrow \text{IR}$ 。

(2) 地址信息的传送

地址信息包含指令地址、顺序执行的后继指令地址、转移地址和操作数地址等 4 类。

① 指令地址——指令地址从 PC 取出,送入 MAR。

地址传输的完整路径是: $PC \rightarrow \text{选择器 A} \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow \text{MAR}$ 。

② 顺序执行的后继指令地址——现行指令地址 $PC+1$,得到后继指令地址,基本的信息路径是: $PC \rightarrow A \rightarrow \text{ALU} (PC+1) \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow PC$ 。

③ 转移地址——按寻址方式形成转移地址,并送入 PC。寻址方式不同,传送路径也不同。

寄存器寻址: $R_i \rightarrow A/B \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow PC$ 。

寄存器间址: $R_i \rightarrow A/B \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow \text{MAR} \rightarrow \text{地址总线} \rightarrow M, M \rightarrow \text{数据总线} \rightarrow \text{MDR} \rightarrow B \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow PC$ 。

④ 操作数地址——按寻址方式形成操作数地址,并送入 MAR。同样,传送路径也因寻址方式而异。

寄存器间址: $R_i \rightarrow A/B \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow \text{MAR}$ 。

变址: 由于形式地址放在紧跟现行指令的下一存储单元中,并由 PC 指示,所以先访存取出形式地址,暂存于 C,再进行变址计算。

取形式地址: $PC \rightarrow A \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow \text{MAR} \rightarrow \text{地址总线} \rightarrow M, M \rightarrow \text{数据总线} \rightarrow \text{MDR} \rightarrow B \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow C$ 。

作变址计算: 变址寄存器 $\rightarrow A \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow \text{MAR}$ 。

$C \rightarrow B \nearrow$

(3) 数据信息的传送

数据信息在寄存器之间、寄存器和主存之间、寄存器和外设之间、主存单元之间以及主存和外设之间传输。

① 寄存器 \rightarrow 寄存器—— $R_i \rightarrow A/B \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow R_j$ 。

② 寄存器 \rightarrow 主存—— $R_i \rightarrow A/B \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow \text{MDR} \rightarrow \text{数据总线} \rightarrow M$ 。

③ 主存 \rightarrow 寄存器—— $M \rightarrow \text{数据总线} \rightarrow \text{MDR} \rightarrow B \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow R_j$ 。

④ 寄存器 \rightarrow 外设—— $R_i \rightarrow A/B \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow \text{MDR} \rightarrow \text{数据总线} \rightarrow \text{I/O 接口}$ 。

⑤ 外设 \rightarrow 寄存器—— $\text{I/O 接口} \rightarrow \text{数据总线} \rightarrow \text{MDR} \rightarrow B \rightarrow \text{ALU} \rightarrow \text{移位器} \rightarrow \text{内总线} \rightarrow R_j$ 。

⑥ 主存 \rightarrow 主存——主存单元间的内容搬迁似乎只需通过 MDR 作为中间缓冲即可,但在寻找目的单元地址时有可能采取间址方式或更复杂的寻址方式,即需要从主存中读取目的地址,并将读得的目的地址经 MDR 传输到 MAR。所以,一般需分成两个阶段实现主存各单元间的传输,先

将读出数据暂存于 C 中，形成目的地址后再将 C 内容经 MDR 写入目的单元。

M (源单元) → 数据总线 → MDR → B → ALU → 移位器 → 内总线 → C, C → A/B → ALU → 移位器 → 内总线 → MDR → 数据总线 → M (目的单元)。

⑦ 主存 ↔ 外设——有两种方式实现主存与外围设备间的数据传输。一种方式是由 CPU 执行通用传输指令，以某种寻址方式指明主存单元与外围接口寄存器的地址，从而实现主存与外围设备间的传输。这样的传输一般以 MDR 为中间缓冲，以便与其他传输指令的执行流程相吻合，即：M ↔ 数据总线 ↔ MDR, MDR ↔ 数据总线 ↔ I/O 接口。

另一种是 DMA 方式，即 CPU 放弃系统总线，由 DMA 控制器控制，通过数据总线实现主存与 I/O 设备之间的直接传输，不动用 CPU 中的寄存器与暂存器，即 M ↔ 数据总线 ↔ I/O 接口。

4. 微命令的设置

在全面分析了各类信息的传输路径后，对指令将如何执行就有了进一步的了解，并为时序的安排与相应微命令的设置打下了基础。以上传送过程包含了两大类操作：内部数据通路操作和外部访存操作。我们分别设置完成这两类操作所需的微命令。

(1) 数据通路操作

图 3-21 所示的数据通路结构可分为 4 段，分段设置有关数据通路操作的微命令如下：

① ALU 输入选择——如微命令 $R_0 \rightarrow A$ (选择 R_0 经 A 门送入 ALU)、 $C \rightarrow B$ (选择 C 经 B 门送入 ALU)、……

② ALU 功能选择——如微命令 $S_3 \sim S_0$ 、M、 C_0 。根据它们的组合 (见本章的 74181 功能表) 选择 ALU 的运算功能。

③ 移位器功能选择——如微命令 DM (直传)、左移、右移。

④ 结果分配——如输入脉冲命令 CPR_0 、CPMAR、……

(2) 访存操作

所需微命令如下：EMAR (地址使能，将 MAR 内容送入地址总线)、R、W (控制读写即数据传送方向)、SMDR、SIR (置入命令)。

在后面拟定指令执行流程后，就可根据指令功能需要选择上述微命令中的有关部分，形成微操作命令序列，实现指令功能。

3.4 运算部件

在计算机中，运算部件主要由输入逻辑、算术逻辑运算部件 (ALU)、输出逻辑三部分组成。其中，ALU 是运算部件的核心，既可以完成算术运算，也可以完成逻辑运算。用硬件实现算术、逻辑运算功能涉及到下面三个问题：如何构成一位二进制加法单元 (全加器)？如何用 n 位全加器 (连同进位信号传送逻辑) 构成一个 n 位并行加法器？如何以加法器为核心，通过输入选择逻辑扩展为具有多种算术、逻辑功能的 ALU？

3.4.1 加法单元

图 3-22 是一位二进制加法单元的示意图，有 3 个输入量：操作数 A_i 、 B_i ，以及低位传来的进位信号 C_{i-1} 。它产生两个输出量：本位和 Σ_i 以及向高位的进位信号 C_i 。这种加法单元考虑了全部 3 个输入量，称为全加器；如果只考虑 2 个输入而不考虑进位，则称为半加器。

全加器的功能是求和，并不需要暂存数据，因而可用门电路构成，如与或门、与或非门或者异或门等。现在广泛采用的求和逻辑形态是：用异或逻辑实现半加，用两次半加实现一位全加，如图 3-23 所示。这种全加器形态的逻辑结构比较简单，有利于实现快速进位传递。

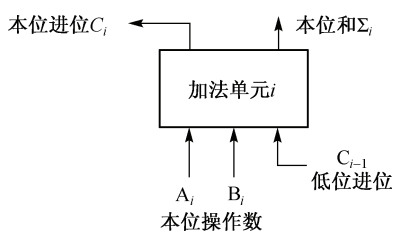


图 3-22 一位加法单元示意图

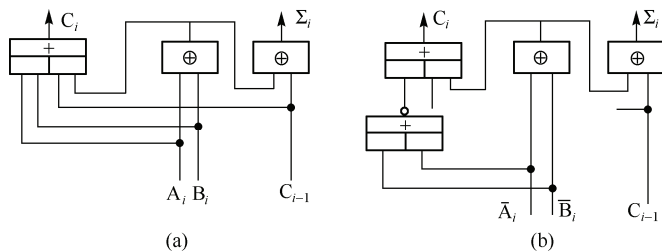


图 3-23 用异或逻辑构成的全加器

在图 3-23(a)中，全加器的输入采用原变量，如 A_i 、 B_i 、 C_{i-1} ，输入与输出之间的关系可以用下面一组逻辑式表示。

$$\Sigma_i = (A_i \oplus B_i) \oplus C_{i-1} \quad (3-1)$$

$$C_i = A_i B_i + (A_i \oplus B_i) C_{i-1} \quad (3-2)$$

式 (3-1) 体现了本位和 Σ_i 与输入之间的关系，它的含义是：第一次半加 $A_i \oplus B_i$ 只考虑了本位的两个输入，第二次半加再考虑低位进位；如果 3 个输入中 1 的个数为奇数，则本位和为 1，否则为 0。

式 (3-2) 体现了本位进位 C_i 与输入之间的关系，表明产生进位的条件有两项：当本位的两个输入 A_i 、 B_i 均为 1 时，不管低位有无进位 C_{i-1} 传来，必然产生进位 C_i ；若 C_{i-1} 为 1，只要 A_i 、 B_i 中有一个为 1，也必然产生进位。其中， $A_i B_i$ 和 $A_i \oplus B_i$ 这两个信号将用来构成进位传递逻辑。

在图 3-23(b)中，全加器的输入采用反变量，如 \bar{A}_i 、 \bar{B}_i ，输入与输出之间的关系可以用另一组逻辑式来表示。

$$\Sigma_i = (\bar{A}_i \oplus \bar{B}_i) \oplus C_{i-1} \quad (3-3)$$

$$C_i = \bar{A}_i \bar{B}_i + (\bar{A}_i \oplus \bar{B}_i) C_{i-1} \quad (3-4)$$

对于一位全加器，采用原变量输入还是反变量输入并无实质上的差异。但在用多个全加器构成一个多位加法器时，为减小信号传递的延时，就需要考虑低位来的进位如何与高一位全加器的极性配合。在构成一个完整的运算器时，还要考虑输入操作数与全加器输入之间的极性配合。为使信号传输的延时尽可能小，全加器逻辑在输入极性和输出极性方面还有多种变化，可以根据不同需要选取不同的逻辑形态。

3.4.2 加法器与进位逻辑

一位全加器只对一位数据求和，如果要将两个多位操作数相加，就需要用全加器构成加法器来实现。加法器分为串行加法器和并行加法器两种，串行加法器因速度太慢，已被淘汰。在现代计算机中，运算器几乎都采用并行加法器，即用多位全加器实现多位数同时相加，所用全加器的位数与操作数位数相同。虽然操作数的各位是同时提供给并行加法器的，但存在进位信号的传递问题，低位运算所产生的进位将会影响高位运算的结果。由于进位传递所经过的门电路级数通常超过每位全加器的门电路级数，因此进位传递的延时会大于全加器自身的延时。加法器的运算速度不仅与全加器的速度有关，更取决于进位传递的速度。

从本质上讲，进位的产生是从低位开始，逐级向高位传播的。进位传递的逻辑结构形态好像链条，因此常将进位传递逻辑称为进位链。并行加法器的逻辑结构包含了全加器单元和进位链两部分。本节先讨论基本的进位链结构，本章 3.4.3 节再结合典型的 ALU 芯片介绍进位链的具体逻辑电路。

1. 进位信号的基本逻辑

如前所述，假定第 $i-1$ 位为低位，则第 i 位产生的进位信号逻辑为：

$$\begin{aligned} C_i &= A_i B_i + (A_i \oplus B_i) C_{i-1} \\ \text{或} \quad C_i &= A_i B_i + (\bar{A}_i \oplus \bar{B}_i) C_{i-1} \\ \text{或} \quad C_i &= A_i B_i + (A_i + B_i) C_{i-1} \end{aligned}$$

我们将上述逻辑用通式表示为：

$$C_i = G_i + P_i C_{i-1} \quad (3-5)$$

$G_i = A_i B_i$ ，称为第 i 位的进位产生函数，或称为本地进位或绝对进位。它的逻辑含义是：若本位的两个输入 A_i 和 B_i 均为 1，必然产生进位，它是不受低位进位传递影响的分量。 P_i 称为进位传递函数，也称为进位传递条件， $P_i C_{i-1}$ 则称为传递进位或者条件进位，且 P_i 可选取三种逻辑形态之一： $(A_i \oplus B_i)$ 或 $(\bar{A}_i \oplus \bar{B}_i)$ 或 $(A_i + B_i)$ ，一般取第一个形态。

由式 (3-5) 表示的进位逻辑，可以归纳出下列两个结论：

① 当本位的两个输入 A_i 和 B_i 中有且仅有一个为 1 时，如果低位有进位传来（即 $C_{i-1}=1$ ），此时 $C_i=0+1=1$ ，故本位必将产生进位。

② 当本位的两个输入 A_i 和 B_i 均为 1，无论低位是否有进位传来（ $C_{i-1}=0$ 或 $C_{i-1}=1$ ），此时都会有 $C_i=1+0=1$ ，故本位也必将产生进位。

对于式 (3-5)，可知当 $P_i=0$ 时低位的进位信号 C_{i-1} 会被屏蔽（因式中的第二项将恒为 0）；而当 $P_i=1$ 时 C_{i-1} 不会被屏蔽。若从进位信号产生条件的角度看，式中的第一项只取决于本位输入 A_i 和 B_i ，而第二项则取决于 A_i 、 B_i 和 C_{i-1} ，这也是把第一项称为“本地进位”而把第二项称为“传递进位”的重要原因。

此外，式 (3-5) 是构成各种进位链结构的基本逻辑式，通过它可以推导出串行进位和并行进位这两种常见的基本形态。

2. 串行进位

串行进位方式是指：逐级地形成各位进位，每一级进位直接依赖于前一级进位，因此也称为行波进位。设 n 位并行加法器中第 1 位为最低位，第 n 位为最高位，初始进位为 C_0 ，则各进位信号的逻辑式如下：

$$\begin{cases} C_1 = G_1 + P_1 C_0 \\ C_2 = G_2 + P_2 C_1 \\ C_3 = G_3 + P_3 C_2 \\ \vdots \\ C_n = G_n + P_n C_{n-1} \end{cases} \quad (3-6)$$

图 3-24 给出了采用串行进位的并行加法器的逻辑结构。在 n 位全加器之间，进位信号采取串联结构，所用元器件较少，但运算时间较长。当每位全加器的两个输入 A_i 、 B_i 中都只有一个为 1，而初始进位 C_0 也为 1 时，加法器的运算时间最长，如 $1010+0101$ ，且 $C_0=1$ 时，进位信号需从第一位开始逐级传递。

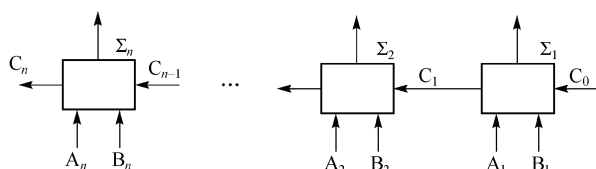


图 3-24 采用串行进位的并行加法器

3. 并行进位

为了提高运算速度，现在广泛采用并行进位结构，即并行地形成各级进位，各进位之间不存在依赖关系，因而这种方式也称为先行进位、同时进位或跳跃进位。在式（3-6）的基础上，采用代入方法，可将每个进位逻辑式中所包含的前一级进位消去，便得到并行进位的逻辑表达式如下：

$$\begin{cases} C_1 = G_1 + P_1 C_0 \\ C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0 \\ C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0 \\ \vdots \\ C_n = G_n + P_n G_{n-1} + \cdots + P_n \cdots P_1 C_0 \end{cases} \quad (3-7)$$

对比式（3-6）和式（3-7），可以发现串行进位与并行进位两种结构的异同。在并行进位结构中，各进位信号是独自形成的，并不直接依赖于前级。当加法运算的有关输入 A_i 、 B_i 和 C_0 稳定后，各位同时形成自己的 G_i 和 P_i ，也就能同时形成各个进位信号 C_i ，从而大大提高了整体的运算速度。

纯并行进位结构在实现时有一个困难，即高位的进位形成逻辑中输入变量增多，这将受到实用器件扇入系数的限制。因此在位数较多的加法器中，常采用分级、分组的进位链结构。

4. 分组进位

典型的分组方法是 4 位一组，组内采用并行进位结构，连同 4 位全加器集成在一块芯片上；组间可以采用串行进位或并行进位，如果采用组间并行，也可将组间并行进位链集成在专用芯片之中；如果加法器位数较长，则可分级构成并行进位逻辑。

下面以组内并行、组间并行的进位链为例，说明这种分级同时进位，或称多重分组跳跃进位的方法。设加法器字长 16 位，每 4 位为一组，分为 4 组，如图 3-25 所示。

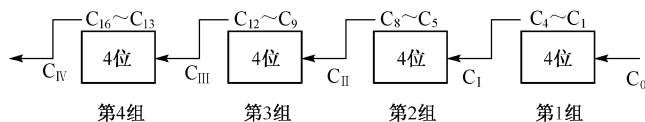


图 3-25 分组进位的逻辑示意图

在这种结构中，初始进位 C_0 送入第 1 组，产生 $C_1 \sim C_4$ ； C_4 是第 1 组的最高进位，又作为第 2 组初始进位送入第 2 组，产生 $C_5 \sim C_8$ 。同理， C_8 送入第 3 组， C_{12} 送入第 4 组。所以 C_4 、 C_8 、 C_{12} 、 C_{16} 是各小组的组间进位，分别用 C_I 、 C_{II} 、 C_{III} 、 C_{IV} 来表示； $C_1 \sim C_3$ 、 $C_5 \sim C_7$ 、 $C_9 \sim C_{11}$ 、 $C_{13} \sim C_{15}$ 则是各小组的组内进位。这样，进位链被分为两级，组内为第一级，组间为第二级，两级都采用并行进位方式。

（1）组内并行进位链

第 1 组的并行进位逻辑如下：

$$\begin{cases} C_1 = G_1 + P_1 C_0 \\ C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0 \\ C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0 \end{cases}$$

C_0 是第 1 组的初始进位。其余各组可照此类推，仅下标序号相应变化。

第 2 组的并行进位逻辑如下：

$$\begin{cases} C_5 = G_5 + P_5 C_1 \\ C_6 = G_6 + P_6 G_5 + P_6 P_5 C_1 \\ C_7 = G_7 + P_7 G_6 + P_7 P_6 G_5 + P_7 P_6 P_5 C_1 \end{cases}$$

如前所述， C_1 是第 1 组产生的组间进位，作为第 2 组的初始进位。

(2) 组间并行进位链

各小组的组间进位信号即是各组产生的最高进位，如

$$C_I = C_4 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0$$

我们令 G_I 、 P_I 分别是第 1 小组的进位产生函数和进位传递函数，并且

$$G_I = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1, \quad P_I = P_4 P_3 P_2 P_1$$

其余各组间进位照此类推，则可得到组间并行进位逻辑：

$$\begin{cases} C_I = G_I + P_I C_0 \\ C_{II} = G_{II} + P_{II} G_I + P_{II} P_I C_0 \\ C_{III} = G_{III} + P_{III} G_{II} + P_{III} P_{II} G_I + P_{III} P_{II} P_I C_0 \\ C_{IV} = G_{IV} + P_{IV} G_{III} + P_{IV} P_{III} G_{II} + P_{IV} P_{III} P_{II} G_I + P_{IV} P_{III} P_{II} P_I C_0 \end{cases}$$

显然，由于各组间进位信号能同时产生，并作为初始进位送往各组的进位输入端，则各小组就能同时产生各组内的进位，大大提高运算速度。

组内、组间两级并行进位链的逻辑形态完全相同。常将组内并行进位链与 4 位全加器集成在一块芯片之中，如 SN74181，另将组间并行进位链单独集成，如 SN74182。

3.4.3 算术逻辑运算部件

利用集成电路技术，可将若干位全加器、并行进位链、输入选择门等三部分集成在一块芯片上，称为多功能算术、逻辑运算部件（ALU）。如 SN74181 就是一种四位片 ALU 芯片，每块芯片上有 4 位全加器、4 位并行进位链、4 位输入选择门，还有 8 位片、16 位片的 ALU 器件。有些参考书中又将这种芯片称为通用函数发生器，即可产生多种输出逻辑函数，具有多种算术运算和逻辑运算功能。用数片这样的 ALU 芯片，加上并行进位链芯片如 SN74182，就可以方便地构成多位 ALU 部件，成为运算部件的核心。

1. ALU 的组成

现以 SN74181 为例讨论它是如何实现多种算术、逻辑运算功能的。

(1) 一位逻辑

如图 3-26 所示，一位 ALU 单元可划分为如下 3 部分。

① 一位加法器——包括由两个半加器构成的一位全加器和由与或非门构成的一位进位门。

② 一位输入选择器——由一对与或非门构成。本位输入两个操作数 A_i 、 B_i 或者 \bar{A}_i 、 \bar{B}_i ，4 个控制信号 S_3 、 S_2 、 S_1 、 S_0 ，可选择 16 种功能。

③ 控制门——用来选择 ALU 做算术运算或逻辑运算。当输入控制信号 $M=0$ 时，开门接收低位来的进位信号 C_{i-1} ，执行算术运算；当 $M=1$ 时，关门不接收 C_{i-1} ，执行逻辑运算，即按位进行运算，与进位无关。

ALU 单元的输入选择逻辑设计得相当巧妙，着眼于构造并行进位链的需要，让选择器输出 X_i 中包含进位传递函数 $P_i = A_i + B_i$ ，而输出 Y_i 中则包含进位产生函数 $G_i = A_i B_i$ 。 S_3 与 S_2 控制选择

左边一个与或非门的输出 X_i , S_1 与 S_0 控制选择右边一个与或非门的输出 Y_i , 逻辑关系见表 3-6。

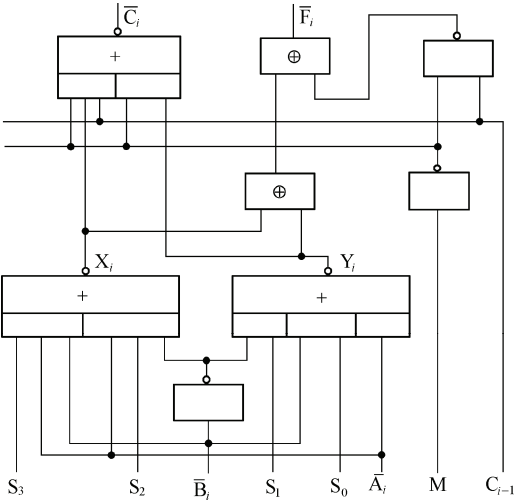


图 3-26 一位 ALU 单元

表 3-6 功能控制信号与选择器输出的对应关系

| S_3 | S_2 | X_i | S_1 | S_0 | Y_i |
|-------|-------|-------------------|-------|-------|-----------------|
| 0 | 0 | 1 | 0 | 0 | A_i |
| 0 | 1 | $A_i + \bar{B}_i$ | 0 | 1 | $A_i B_i$ |
| 1 | 0 | $A_i + B_i$ | 1 | 0 | $A_i \bar{B}_i$ |
| 1 | 1 | A_i | 1 | 1 | 0 |

选择不同的控制信号 $S_3S_2S_1S_0$, 可获得不同的输出 F_i , 从而实现不同的运算功能。这就是我们一再强调的分析线索：从信息传输的角度理解运算器的组成原理，通过不同的输入选择，实现不同的运算功能。

(2) 多位逻辑

图 3-27 给出了 4 位 ALU 芯片 SN74181 的逻辑组成，包含以下几部分。

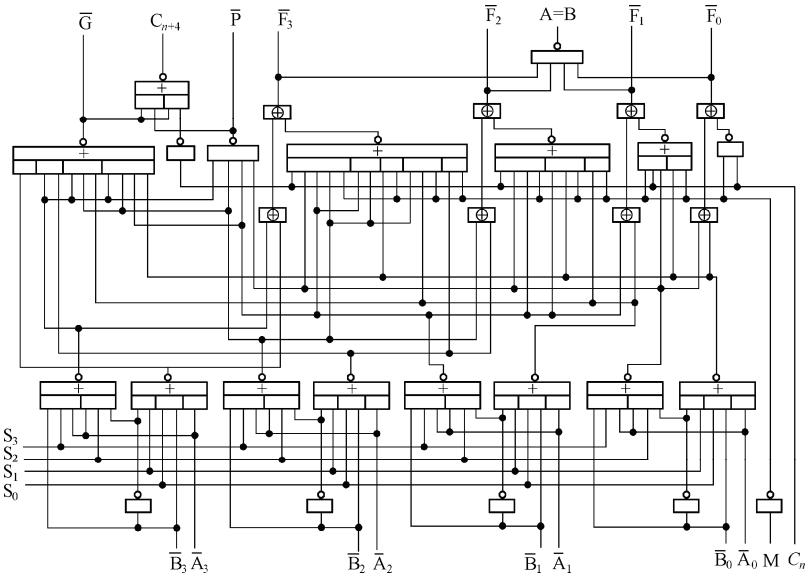


图 3-27 SN74181 内部逻辑结构

① 4 位 ALU。每一位的逻辑与图 3-26 相同，只是 4 位公用一个控制门，位于图的右下角，由 M 控制做算术运算或逻辑运算。4 位全加器位于图的上半部。4 位输入选择逻辑位于图的下半部，它们根据输入控制信号 $S_3S_2S_1S_0$ 的不同状态，向全加器提供操作数的多种输入组合，选择运算功能。

② 组内并行进位链。片内 4 位为一小组，组内采用并行进位结构。初始进位输入 C_n 。对外输出 3 个信号：进位 C_{n+4} 、小组进位产生函数 \bar{G} 、小组进位传递函数 \bar{P} 。利用 C_{n+4} 可以构成组间串行进位，将 \bar{G} 和 \bar{P} 送往 SN74182 可构成组间并行进位。

分组后组内并行且组间并行进位的相关的逻辑式如下：

$$\begin{cases} C_{n+1} = G_0 + P_0C_n \\ C_{n+2} = G_1 + P_1G_0 + P_1P_0C_n \\ C_{n+3} = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_n \\ C_{n+4} = G + PC_n = \bar{G}\bar{P} + \bar{G}\bar{C}_n \end{cases}$$

其中， $G = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$ ， $P = P_3P_2P_1P_0$ ，相应的逻辑电路位于图 3-27 的上半部。

③ 符合比较“ $A=B$ ”：SN74181 可执行异或运算，输出 $A \oplus B$ 或 $\overline{A \oplus B}$ ，通过输出门“ $A=B$ ”可获得比较结果。该输出门位于图的最上端。

SN74181 的外特征，即引脚框图示于图 3-28 中。从该图可以清晰地看出，74181 的使用可能有两种高低电平极性关系。

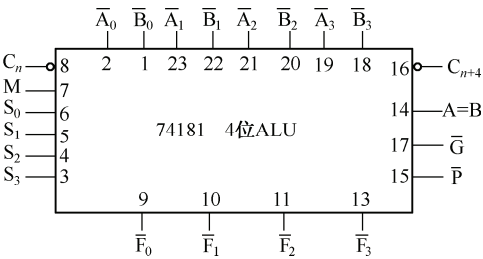


图 3-28 SN74181 引脚框图

⊙ 输入和输出均为反变量。输入：操作数 $\bar{A}_3 \sim \bar{A}_0$ 、 $\bar{B}_3 \sim \bar{B}_0$ ，进位 C_n ，控制信号 M、 S_3 、 S_2 、 S_1 、 S_0 ；输出：运算结果 $\bar{F}_3 \sim \bar{F}_0$ ，进位 C_{n+4} 、 \bar{G} 、 \bar{P} ，符合 $A=B$ 。

⊙ 输入和输出均为原变量。输入：操作数 $A_3 \sim A_0$ 、 $B_3 \sim B_0$ ，进位 \bar{C}_n ，控制信号 M、 S_3 、 S_2 、 S_1 、 S_0 ；输出：运算结果 $F_3 \sim F_0$ ，进位 \bar{C}_{n+4} 、G、P，符合 $A=B$ 。

2. ALU 的运算功能

表 3-7 概括了 SN74181 可能实现的各种算术运算与逻辑运算功能，及其对应的控制信号状态。一旦确定 ALU 所需执行的运算功能，就可根据表 3-7 选择控制信号的相应组合。从运算器与控制器的界面看，由 74181 构成的 ALU 属于运算器范畴，而 M、 $S_3S_2S_1S_0$ 等则由控制器产生。注意，表 3-7 中的运算符“+”是逻辑加（即逻辑或），为了区别，算术加用中文“加”字表示。

表 3-7 SN74181 功能表

| 工作方式选择位 ($S_3S_2S_1S_0$) | 运算模式位 (M) | |
|----------------------------|--------------------------|------------------------|
| | 逻辑运算 (M=1) | 算术运算 (M=0) |
| 0000 | \bar{A} | A 减 1 |
| 0001 | \overline{AB} | AB 减 1 |
| 0010 | $\bar{A} + B$ | $A\bar{B}$ 减 1 |
| 0011 | 逻辑 1 | 全 1 |
| 0100 | $A + \bar{B}$ | A 加 ($A + \bar{B}$) |
| 0101 | \bar{B} | AB 加 ($A + \bar{B}$) |
| 0110 | $\bar{A} \oplus \bar{B}$ | A 加 \bar{B} |
| 0111 | $A + \bar{B}$ | $A + \bar{B}$ |
| 1000 | $\bar{A}B$ | A 加(A+B) |

续表

| 工作方式选择位 ($S_3S_2S_1S_0$) | 运算模式位 (M) | |
|----------------------------|----------------|----------------------|
| | 逻辑运算 ($M=1$) | 算术运算 ($M=0$) |
| 1001 | $A \oplus B$ | A 加 B |
| 1010 | B | $A\bar{B}$ 加 $(A+B)$ |
| 1011 | $A+B$ | $A+B$ |
| 1100 | 逻辑 0 | 0 |
| 1101 | $A\bar{B}$ | AB 加 A |
| 1110 | AB | $A\bar{B}$ 加 A |
| 1111 | A | A |

从设计方法看,我们可以选择两种思路来设计 ALU 部件的结构逻辑。一种是先确定所需实现的运算功能,再通过常规的逻辑设计方法,即真值表、卡诺图化简等,得到逻辑表达式,然后画出逻辑电路图。由于输入变量多,除 A_i 、 B_i 、 C_n 外,还有 M 、 $S_3S_2S_1S_0$,化简很困难,因此 74181 的设计选择了另一种思路:先选取恰当的输入选择组合,即 X_i 、 Y_i 对 $S_3S_2S_1S_0$ 的逻辑关系,再导出可能实现的 16 种算术运算和 16 种逻辑运算功能。其中包含了我们需要的基本运算功能,如 $A+B$ 、 $A-B$ (利用 $A+\bar{B}$)、 A 加 1 ($M=0$ 、 $S_3S_2S_1S_0=1111$ 、 $C_n=1$)、 A 减 1、逻辑与 AB 、逻辑或 $A+B$ 、求反 \bar{A} 或 \bar{B} 、异或 $A \oplus B$ 、传输 A (即输出 A, $S_3S_2S_1S_0=1111$)、传输 B ($M=1$ 、 $S_3S_2S_1S_0=1010$)、输出 0、输出 1 等。

3. ALU 的进位逻辑

用若干片 SN74181 可以方便地构成更多位数的 ALU 部件。片内已实现组内并行进位,如果采取组间串行进位结构,只需将几片 SN74181 简单级联,即将各片的进位输出 C_{n+4} 送往高位芯片的进位输入端 C_n ,如图 3-29 所示。



图 3-29 组间串行进位的 ALU

如果采用组间并行进位结构,需增加并行进位链芯片 SN74182。图 3-30 提供了一个 16 位 ALU 连接实例。74181 输出的小组进位产生函数 G 和进位传递函数 P ,可作为并行进位链 74182 的输入,而 74182 向各 74181 提供组间进位信号。74182 的输出 (图中的 \bar{G} 、 \bar{P}) 还可支持更高一级的并行进位链,从而可构造更长位数的 ALU。

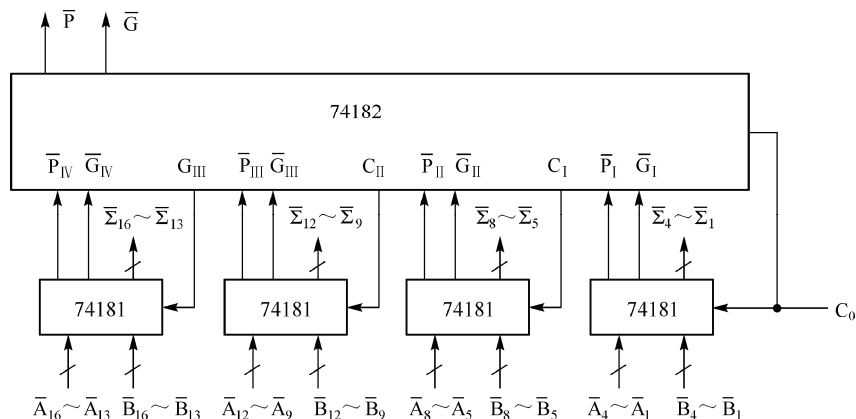


图 3-30 组间并行进位的 ALU 结构

3.4.4 运算器的组织

基本的运算器组织包含如下逻辑组成：实现基本算术逻辑运算功能的 ALU、提供操作数和暂存运算结果的寄存器组、有关的判别逻辑（如结果是否为 0？为正/负？有无进位？有无溢出？等等），或加上局部控制电路。将这些功能模块连接成一个整体时，需要解决两个问题。其一，如何向 ALU 提供操作数？一种方法是在 ALU 输入端加多路选择器，另一种方法是在 ALU 输入端加一级锁存器（暂存器）。其二，寄存器组采用什么样的结构？一种方案是采取独立寄存器结构，可以同时向 ALU 提供两个操作数，缺点是集成度低；另一种方案是采取小型存储器结构，缺点是每次只能提供一个输入，当然，采用双口存储器可以在一定程度上弥补这一不足。

1. 具有多路选择器的运算器

如图 3-31 所示，CPU 内部总线是一组单向传送的数据线，它将运算结果送往各寄存器。寄存器组是一组彼此在逻辑上独立的寄存器，它们有各自的输入端和输出端。如果向某个寄存器发送同步打入脉冲，则可将内部总线上的数据送入该寄存器；如果同时发几个输入脉冲，则可将总线上的同一数据同时送入几个相关的寄存器。各寄存器可以独立、多路地将数据送至 ALU 输入端的多路选择器，因而 ALU 可同时获得两路数据输入。多路选择器既可实现操作数选择，也可通过输入选择实现功能扩展。

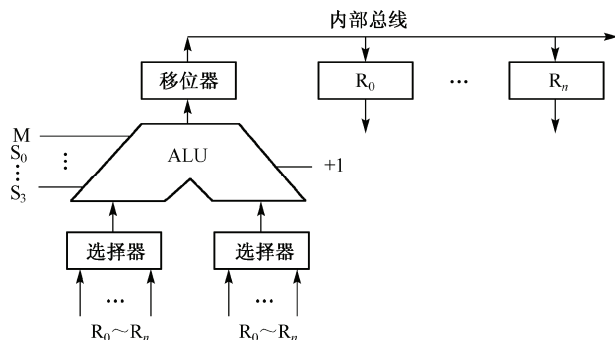


图 3-31 具有多路选择器的运算器组成

ALU 可采取如图 3-27 所示的结构，通过其控制端 M、 $S_3S_2S_1S_0$ 、末端初始进位“+1”等，可实现 ALU 本身的功能选择。ALU 输出经移位器实现移位选择：直传不移位、左移、右移等，移位器常用多路选择器通过斜传实现移位。这种组成模型较为简单，因此本教材中设计的模型机，其运算部件也采用了上述的结构。

2. 具有输入锁存器的运算器

如图 3-32 所示，CPU 内部总线是一组双向传送的数据线，而寄存器组采用小规模高速存储器结构，每次由控制器发命令可选中某一单元，相当于选中一个寄存器进行读或写，也就是说，每次只能向 ALU 提供一个操作数。为了对双操作数进行运算操作，就需要在 ALU 输入端前设置一级锁存器，用来暂存操作数。例如，要实现 $R_0+R_1 \rightarrow R_0$ 操作，可通过内部总线先将 R_0 的数据送入锁存器 1，再通过内部总线将 R_1 的数据送入锁存器 2，或直接送加法器，然后相加，并将结果经内部总线送入 R_0 。

寄存器组中也可包含若干暂存器，用来与系统总线相连接。例如，通过系统总线访问主存获得的数据可先暂存于暂存器之中，再通过内部总线送入 ALU；运算结果也可先暂存于暂存器中，再通过系统总线送入主存储器。具体的结构有多种变化，可结合具体机型分析。

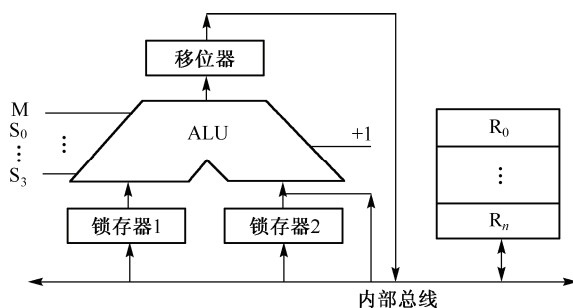


图 3-32 具有输入锁存器的运算器组成

3. 位片式运算器

采用大规模集成电路技术可将 n 位寄存器组、 n 位选择器、 n 位 ALU、 n 位移位器等集成在一块芯片上，成为一片 n 位运算器。将若干块这样的位片连接起来，就能构成较长位数的运算器，这已成为一种实用的设计方法。这种方法使系统组成灵活方便，并可大批量生产位片。代表性的位片有 AMD 2900/29000/29300 系列。图 3-33 是 AMD 2900 系列位片的粗框结构，图中略去了一些细节。

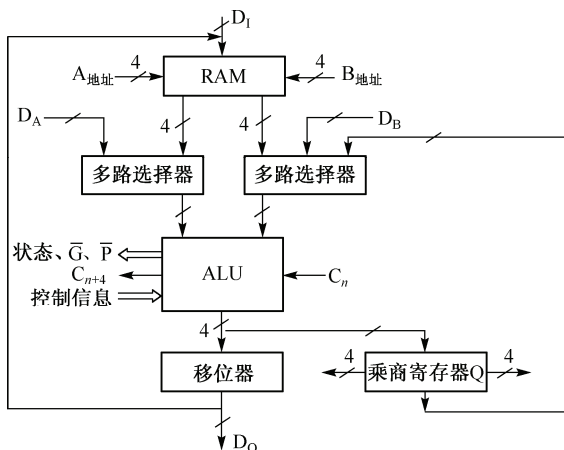


图 3-33 AMD 2900 系列位片的组织结构

双口随机存储器 (RAM) 构成了一个 16×4 位的通用寄存器组，它有 16 个单元，相当于 16 个寄存器，每个寄存器 4 位。所谓双端口，是指可同时向它送入两个地址：A 地址和 B 地址，因而可同时选中两个寄存器，它们同时将各自的 4 位数据送往多路选择器，供 ALU 运算处理。

ALU 类似于 SN74181 的逻辑结构，在此基础上进一步扩展了功能，可实现乘、除运算。它的功能控制信号有 M 、 $S_3S_2S_1S_0$ 等，进位输入 C_n ，进位输出 C_{n+4} ，进位辅助函数 \bar{G} 、 \bar{P} 等，还输出某些状态信息。

乘商寄存器 Q 用于乘、除运算。在乘法运算时用来存放乘数，运算结束时存放乘积的低位部分；在除法运算时用来存放商。Q 寄存器也可作为辅助寄存器使用。

多路选择器实现 ALU 的输入选择，它的信息来源有：通用寄存器组、外部直接输入 D_A 和 D_B 、乘商寄存器 Q。

D_1 、 D_0 分别是位片的数据输入、输出端。虽然每片只有 4 位，但将若干位片拼接起来，再加上微程序控制器芯片，就可以方便地构成 CPU。

3.5 组合逻辑控制方式

在完成模型机的总体设计之后，我们将进一步设计模型机的控制器逻辑。本节先采用组合逻辑控制方式进行设计，包括安排时序、拟定指令流程与微命令序列、形成控制逻辑等，3.6 节再讨论如何采用微程序控制方式进行设计。指令流程和相应微命令序列的拟定主要取决于数据通路结构，所以两种控制器在这一部分的设计有不少内容是相似的，其区别主要在时序划分以及最后形成微命令的方式上。

采用组合逻辑控制方式来产生所需的微命令的控制器称为组合逻辑控制器（Combinational Logic Controller）。那么，什么是组合逻辑控制方式呢？我们知道，每个微命令的产生都需要逻辑和时间条件，将这些条件作为输入信号，微命令作为输出，它们之间的关系可用逻辑式来表示，也就可以用组合逻辑电路来实现。每种微命令都需要一组逻辑电路，全机所有微命令所需的逻辑电路经过组合优化后就构成了微命令发生器。执行指令时，由组合逻辑电路（微命令发生器）在相应条件下发出所需的微命令，控制有关操作。这种产生微命令的方式就是组合逻辑控制方式。在形成逻辑电路之前，还需对产生微命令的条件进行综合、优化，公用某些逻辑变量或中间逻辑函数，以便使逻辑式尽可能简单，减少微命令发生器所用的元器件数和逻辑门数，提高产生微命令的速率。在控制器制造完之后，这些逻辑电路之间的连接关系就固定下来，不易改动，因而组合逻辑控制器在有些技术资料中又称为硬连线控制器。在这种控制器中，微命令形成逻辑的元器件数量在控制器中占较大比重。

图 3-34 是一种组合逻辑控制器的原理框图，主要包括微命令发生器、指令寄存器 IR、程序计数器 PC、状态字寄存器 PSW、时序系统等部件。

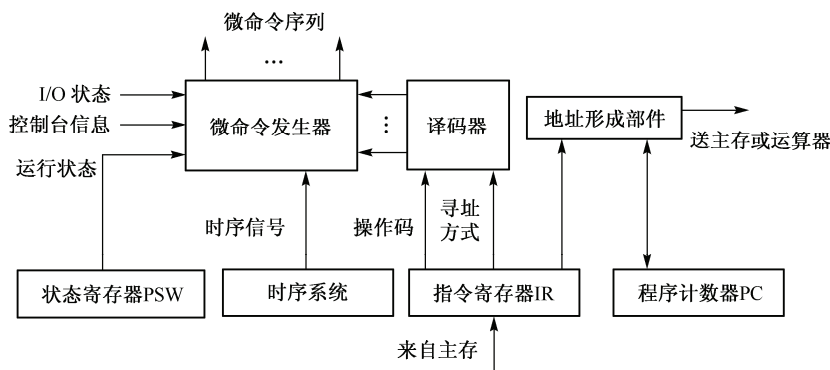


图 3-34 组合逻辑控制器的原理框图

微命令发生器是由若干门电路组成的组合逻辑电路。从主存读取到的现行指令存放在 IR 中，其中，操作码与寻址方式代码分别经译码电路形成一些中间逻辑信号，送入微命令发生器，作为产生微命令的基本逻辑依据。微命令的形成还需考虑各种状态信息，如 PSW 所反映的 CPU 内部运行状态、由控制台（如键盘）产生的操作员控制命令、I/O 设备与接口的有关状态、外部请求等。微命令是分时产生的，所以还需引入时序系统提供的周期、节拍、脉冲等时序信号。IR 中的地址段信息送往地址形成部件，按照寻址方式码形成实际地址，或送主存以访问主存单元；或送往运算器，按指定的寄存器号选取相应的寄存器。当程序顺序执行时，PC 增量计数，形成后续指令的地址；当程序需要转移时，IR 中的地址段信息经地址形成部件产生转移地址，送入 PC，使程序发生转移。

3.5.1 组合逻辑控制器时序系统

组合逻辑控制器依靠不同的时间标志,使 CPU 分步工作,模型机按常规采用工作周期、时钟周期、工作脉冲三级时序。也就是说,将一条指令从读取到执行完(即一个指令周期),按不同的操作阶段划分为若干工作周期(也叫机器周期);每个工作周期又按不同的分步操作划分为若干时钟周期(也叫节拍);在每个时钟周期中,再按所需的定时操作设置相应的工作脉冲。

1. 工作周期

根据各类指令的需要和一些特殊工作状态的需要,模型机设置了 6 种工作周期状态,用 6 个周期状态触发器作为它们的标志。其中,4 个工作周期(取指、源、目的、执行)用于指令的正常执行,2 个工作周期(中断、DMA)用于 I/O 传输控制。某一时期内只有其中一个工作周期状态触发器的输出信号为 1,指明 CPU 现在所处的工作周期状态,为该阶段的工作提供时间标志与依据。

① 取指周期 FT——取指所需的操作安排在 FT 中完成,包括将指令从主存取出并送入 IR、修改 PC 等操作,这是每条指令都必须经历的。在 FT 中完成的操作是与指令操作码无关的公共性操作,但取指周期结束后将转向哪个工作周期,则与 FT 中取出的指令类型及所采用的寻址方式有关。

② 源周期 ST——如果需要从主存中读取源操作数(非寄存器寻址),则进入 ST。在 ST 中将依据指令寄存器 IR 的源地址段信息进行操作,形成源地址,读取源操作数,将其暂存于 C。

③ 目的周期 DT——如果需要从主存中读取目的地址或目的操作数(非寄存器寻址),则进入 DT。在 DT 中,将依据指令寄存器 IR 的目的地址段信息进行操作,形成目的地址放在 MAR 中,或读取目的操作数暂存于 D。

④ 执行周期 ET——取得操作数后,CPU 进入 ET,这也是各类指令都需经历的最后一个工作阶段。在 ET 中将依据 IR 中的操作码执行相应的操作,如数据传输、算术运算、逻辑运算、保存返回地址、获得转移地址等。在 ET 中还要将后继指令的地址(顺序执行或转移)送入到 MAR 中,为下一指令周期读取新指令做好准备。

⑤ 中断周期 IT——除了考虑指令的正常执行,还需考虑外部请求带来的变化。在响应中断请求之后,到执行中断服务程序之前,需要一个过渡期,称为中断周期 IT。在 IT 中,将直接依靠硬件进行关中断、保存断点、转服务程序入口等操作。

⑥ DMA 周期 DMAT——响应 DMA 请求之后,CPU 进入 DMAT。在 DMAT 中,CPU 交出系统总线的控制权,即 MAR、MDR 与系统总线断开(呈高阻态),改由 DMA 控制器控制系统总线,实现主存与外围设备间的数据直传。因此对 CPU 来说,DMAT 是一个空操作周期(只进行周期切换和 DMA 初始化等)。

各工作周期状态之间的转换示于图 3-35 中。

FT 结束后,对于双操作数指令,如果数均在主存中,则进入 ST,之后进入 DT、ET;如果数均在寄存器中,则进入 ET。对于单操作数指令,如果数在主存中,则进入 DT、ET;如果数在寄存器中,同样进入 ET。对于转移指令,FT 结束后直接进入 ET。因此,在每个周期结束前,都要判断下一周期的状态将是什么,并为此准备好进入下一周期的条件。

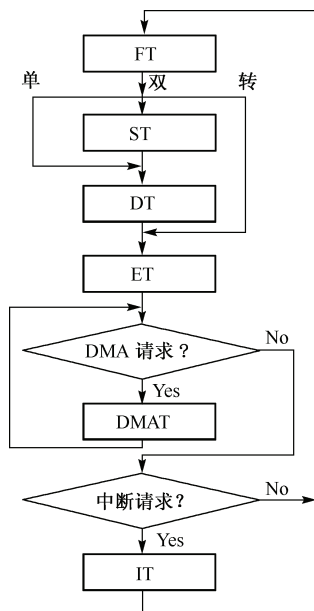


图 3-35 CPU 控制流程

到本周期结束的时刻，再实现周期状态的定时切换。

由于 DMA 周期要实现的是高速数据直传，所以让 DMA 请求的优先级高于中断请求。因而在一条指令将要结束时，先判断有无 DMA 请求，若有请求，将插入 DMAT。请注意，实际的计算机大多允许在一个系统总线周期结束时就插入 DMAT。本模型机为了简化其控制逻辑，限制在一条指令结束时才判别与响应 DMA 请求。控制流程还表明：如果在一个 DMAT 结束前又提出了新的 DMA 请求，则允许连续安排若干 DMA 周期。

如果没有 DMA 请求，则判断有无中断请求，若有，则进入 IT，在时间上这两种判断可同时进行，只不过在逻辑上要优先 DMA 请求而已。在 IT 中完成必要的过渡期工作后，将转向新的 FT，开始中断服务程序的执行。

如果也没有中断请求，那就返回 FT，从主存读取后继指令。

2. 时钟周期（节拍）

指令的读取与执行既有 CPU 内部数据通路操作，也包含访问主存的操作。为简化时序控制，模型机将两类操作周期统一起来，即以主存访问周期所需时间为时钟周期的宽度，这里设置为 $1\mu\text{s}$ 。当然，这对于 CPU 内部操作来说，在时间上是比较浪费的。

一个工作周期包含若干时钟周期，根据不同指令的需要，时钟周期数可变。为此设置了一个时钟周期计数器 T ，其计数循环可随需要而变。若本工作周期应当结束，则发命令 $T=0$ ，计数器 T 复位，从 $T=0$ 开始一个新的计数循环，进入新的工作周期。若本工作周期还需延长，则发命令 $T+1$ ，计数器 T 将继续计数，出现新的节拍。计数器 T 的计数值经译码产生时钟周期状态，如 T_0 、 T_1 、 T_2 等，作为分步操作的时间标志，如图 3-36 所示。

3. 工作脉冲

时钟周期表示一个时间段，在这段时间内可以进行某种数据通路操作，如两数相加。但有些操作需要同步定时脉冲进行控制，如将稳定的运算结果打入寄存器，或者进行周期状态切换。模型机在每个时钟周期的末尾发一个工作脉冲 P ，作为各种同步脉冲的来源，如图 3-37 所示。在实际应用中，可以直接用时钟信号的上升/下降沿代替工作脉冲。

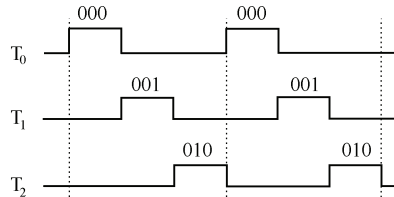


图 3-36 T 计数值译码产生时钟周期

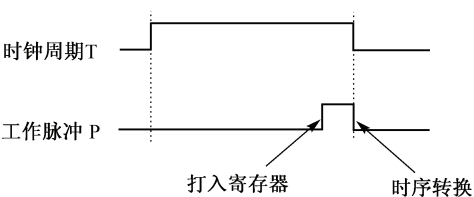


图 3-37 时钟周期与工作脉冲的关系

工作脉冲 P 的前沿作为打入寄存器的定时，标志着一次数据通路操作的完成。 P 的后沿作为时序转换的定时，此刻如果本工作周期未结束，则对时钟周期计数器 T 计数，进入新的节拍；如果本工作周期结束，则将 T 清零，并清除本工作周期状态标志，重新设置新的工作周期状态标志。

3.5.2 指令流程与操作时间表

控制器设计的核心是拟定指令流程与形成微命令序列。因此，本节是模型机设计的关键一步，也是全章的一个重点，通过有关讨论应能较深入地了解 CPU 执行指令的工作机制。我们将分成两个层次进行讨论：

- ⊙ 在寄存器传输级，拟定各类指令的执行流程，也就是确定指令执行的具体步骤，即各类信息如何分步地按要求流动。
- ⊙ 拟定操作时间表，即给出实现上述流程所需的微操作命令序列。其中包含维持一个时钟周期的电位型微命令，以及短暂的脉冲型微命令。操作时间表还将表明出现各种微命令的逻辑条件与时间条件。

通过这两种层次的工程化描述，我们就清晰地确定 CPU 在什么时间、根据什么条件、发出什么命令、做什么事。

有两种可供选择的设计线索：一种方法是以工作周期为线索，按工作周期分别拟定各类指令在本工作周期内的操作流程，再以操作时间表的形式分时列出应发出的微命令及逻辑条件。这种方法的优点是便于综合与化简微命令的逻辑式，容易取得比较优化的结果。另一种方法是以指令为线索，按指令类型分别拟定操作流程。这种方法的优点是对指令的执行过程拟出了清晰的线索，便于理解 CPU 的工作过程。模型机设计采用后一种方法，再插入中断周期和 DMA 周期的流程。由于取指周期是各类指令流程都需首先经历的，与指令类型无关，所以我们先拟定取指周期流程，再分别拟定各类指令流程。

在指令执行过程中，必须要准确知道 CPU 当前处于哪一个工作周期中，才能发出恰当的控制命令。如前所述，控制器中分别设置了 6 个与工作周期一一对应的状态触发器，通过触发器的输出信号来标识不同的工作周期。

1. 取指周期 FT

指令在执行之前都是放在存储器中的，任何指令的执行都必须先进入取指周期去取出指令。CPU 如何进入取指周期呢？

(1) 进入 FT 的方式和条件

在初始化和运算过程中，有 5 种情况需要进入取指周期，可分别采用置入方式或同步打入方式，使取指周期状态触发器 FT 为 1，如图 3-38 所示。

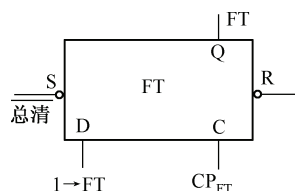


图 3-38 取指周期状态触发器

① 初始化置入 FT

- ⊙ 上电初始化：当机器加电后，系统首先产生一个总清信号，维持若干毫秒（低电平有效），使有关部件进入正确的初始化状态，包括使 FT 为 1，让 CPU 从取指周期开始正常地执行指令。由于此时的时序系统尚未正常（也需要一个初始化阶段），所以采取由 S 端置入的方法使 FT 为 1。
- ⊙ 复位初始化：按下复位键后，也会产生一个强制初始化过程，情况与上电初始化相似，所以可让复位键也产生总清信号。由于这是在封锁时钟的状态下预置全机初始状态，所以也采用 S 端置入方式。抬起复位键后，开放时钟，CPU 将开始取指操作，进入监控程序运行状态，再通过监控程序实现进一步的初始化。

② 运行过程中同步打入 FT

在正常的程序运行过程中，可用同步方式实现周期状态转换。若要进入 FT，则事先在状态触发器 D 端准备好条件 1→FT（电位型微命令，高电平有效），然后用脉冲 P 的后沿，即 CP_{FT} 的上升边沿将 1 打入 FT。若要结束 FT 状态，则在 D 端不加 1→FT 命令，让 D 端电平为 0，脉冲 P 的后沿将 0 打入 FT，使 FT 变为 0，表示取指周期结束。

分析图 3-35，CPU 控制流程有三种情况将进入新的取指周期，需先准备好 1→FT 条件。在执

行周期 ET 中（即现行指令将执行完毕时），如果不响应 DMA 请求与中断请求，主程序正常执行，应进入下一条指令的 FT；在中断周期这一过渡阶段结束后，应转入中断服务程序，即进入中断服务程序第一条指令的 FT；在 DMA 周期完成一次 DMA 传送后，如果没有新的 DMA 请求或中断请求需要响应，则应恢复执行被暂停的主程序，因此也应进入主程序返回后第一条指令的 FT。

由此，可获得 $1 \rightarrow FT$ 命令的逻辑式如下：

$$1 \rightarrow FT = ET(\overline{1 \rightarrow DMAT} \cdot \overline{1 \rightarrow IT}) + IT + DMAT(\overline{1 \rightarrow DMAT} \cdot \overline{1 \rightarrow IT})$$

其他 5 个周期状态的建立与切换逻辑也可参照这种办法形成。

(2) 取指流程

图 3-39 以寄存器级传送语句（如 $M \rightarrow IR$ ）的形式描述取指周期内的各操作步骤，共占用一个时钟周期（节拍）。在这一节拍内 CPU 同时执行两项操作：第一项是从主存中读取指令，代码有效时即可置入指令寄存器 IR（假定前面已准备好了指令地址）。第二项是修改程序计数器 PC 的内容，让 $PC+1$ ，则修改后的 PC 指向紧跟现行指令的下一单元。为什么这两项操作可以同时进行呢？因为它们各自使用的数据通路没有冲突，读取指令经由数据总线，而 $PC+1$ 经由

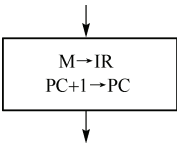


图 3-39 取指流程

它们各自使用的数据通路没有冲突，读取指令经由数据总线，而 $PC+1$ 经由 ALU 与内部总线。因此，在拟定指令流程时，从优化操作步骤的角度考虑，凡是数据通路没有冲突、时间上也不矛盾的操作，都可以安排在一个时钟周期内并行执行。

(3) 操作时间表

表 3-8 是取指周期的操作时间表，给出了为实现取指流程所需的微命令序列。微命令是最基本的控制命令，分为电位型和脉冲型两类。表的左栏是工作周期状态与节拍序号综合标注，如 FT0。中栏给出了在本拍中应发的电位型微命令，这些微命令按序发出，共维持一个时钟周期。有些微命令只在某些逻辑条件下才发出，则进一步在括号中标注其补充逻辑条件。如果表中不便写全，或者在拟定本表时还不能给出全部逻辑条件，要等到全部指令流程和操作时间表都列完后才能全部确定，则可在括号中先注明逻辑式序号，以后再补充相应逻辑式。右栏给出本拍中应发的脉冲型微命令，并示意性地表明脉冲是在时钟周期的末尾才发出的，由工作脉冲 P 或其反相脉冲（负脉冲） \bar{P} 同步定时。

表 3-8 取指周期的操作时间表

| | | 电位型微命令 | 脉冲型微命令 | |
|-----|------|--|--------|--|
| FT0 | 按序发出 | EMAR R SIR PC→A S ₃ $\bar{S}_2\bar{S}_1\bar{S}_0\bar{M}C_0$ DM | 无 | 无 |
| | 三者之一 | 1→ST（逻辑式 1） 1→DT（逻辑式 2） 1→ET（逻辑式 3） | 同时发出 | CPPC CPFT(\bar{P}) CPST(\bar{P}) CPDT(\bar{P}) CPET(\bar{P}) CPT(\bar{P}) |

CPU 可能需要成百上千的微命令，操作时间表中只列出其中在本节拍内有效的微命令，无效的微命令或 0 电平则不必列出。受表中微命令控制的有 3 类操作：访存操作、CPU 内部数据通路操作、时序转换操作。下面分析 FT 操作时间表中各微命令的含义。

① 控制访存操作的微命令——地址使能命令 EMAR 使 MAR 输出有效，经地址总线送往主存。读令 R 送至存储芯片，读出指令。置入命令 SIR 是 IR 置入的开门命令，若读出信息为 1，则 IR 中对应位被置为 1。只有在 FT 中才发出 SIR，将读出的指令代码直接送往 IR。其他时间读出的内容将送入 MDR。

② 控制 CPU 内部数据通路操作的微命令——PC→A 是一个选择命令，使多路选择器 A 选择 PC 送入 ALU，封锁 B 选择器。C0 为 1 则以初始进位形式提供 1。控制信号 S₃、 \bar{S}_2 、 \bar{S}_1 、S₀、 \bar{M} 控制 ALU 功能为带进位加（A+B），但 B=0，所以实际上是实现 A 加 1 功能。DM 是对移位器发

出的直传命令。CPPC 是打入 PC 的同步定时命令，只有当该脉冲的前沿到来时，PC 内容才能被修改。

③ 控制时序切换的微命令——由于 FT 只占一个时钟周期，所以完成 FT 操作之后就应当进入新的工作周期，节拍状态又从 0 开始计数，即维持 T_0 不变。根据在 FT 中读取的指令，决定进入 ST 或 DT，或直接进入 ET。因此操作时间表中列出了 3 种可能建立的状态：1→ST 或 1→DT 或 1→ET，在括号中注明相应的逻辑条件。现在暂时无法写明这些逻辑条件，暂以逻辑式 1~3 标注，待确定全部指令流程后再补充。在 FT 末尾同时发出 4 个打入脉冲 CPFT、CPST、CPDT、CPET，以 \bar{P} 脉冲同步定时，P 与 \bar{P} 的后沿位于相同时刻，且后者的后沿是上升沿，符合 D 触发器的要求。由于表中未发 1→FT（即触发器 D 端为 0），而 1→ST~1→ET 中只有一个为 1，因此 FT 结束后将只有一个工作周期状态触发器为 1，其余 5 个均为 0，实现了周期的切换。表中未发计数器的计数命令，即 $T+1=0$ ，所以尽管发出了 CPT，也不会改变 FT_0 的状态。只有发出了 $T+1$ 命令和 CPT (\bar{P}) 时，才能使节拍计数器加 1。

至此我们以取指周期为例，阐述了两个层次上的工作机理，即用寄存器级传送语句描述指令流程，以操作时间表形式描述具体的微命令序列（以后的叙述将会粗略一些）。

2. MOV 指令

如果现行指令是 MOV 指令，CPU 将执行 MOV 指令流程，如图 3-40 所示。对于已阐述过的 FT 流程，图中不再细述。

MOV 指令的流程图中包含了各种寻址方式的组合，流程分支的逻辑依据就是寻址方式字段的编码，图中标注为相应的助记符（汇编符号）。每个工作周期结束时要判断后继工作周期是哪一个周期。通过对 MOV 指令流程的剖析，能够了解各种寻址方式的具体实现过程，因此是剖析整个指令系统执行流程的突破口。为了方便学习和理解，我们将指令流程安排得非常规整，寻址方式由简到繁，在同一种节拍中尽量使逻辑依据相近、操作内容相近。但在时间安排上就不一定优化，如有的操作可以在一步中完成，而按照上述考虑，在流程中却分为两步来做。下面对 MOV 指令流程进行分析。

（1）取指周期 FT

所有指令的公共操作。注意：在 FT 结束时将根据源寻址方式作出判别与分支，决定是否进入 ST。如果源寻址方式为非寄存器寻址，则进入 ST。

（2）源周期 ST

在 ST 中，首先根据源寻址方式决定在 ST 中的分支。

① R 型。源操作数在指定寄存器中，将来在 ET 中可直接送往 ALU。因此，可暂时不取源操作数，也就不经历 ST。注意：寄存器寻址不进入 ST，但考虑到与其他寻址方式统一处理，在流程图中用一条垂线表示。

② (R)型。第一拍从指定寄存器 R_i 中取得地址，第二拍访存读取操作数，经 MDR 送入 C 中暂存。在 ST 中需要暂存的信息，一般都暂存于 C 中。

③ -(R)型。第一拍先修改地址指针内容，即指定寄存器 R_i 的内容减 1，所得结果同时打入 R_i 与 MAR，形成源地址。第二拍访存读取操作数，送入 C 暂存。

④ I(R)+型。第一拍取得地址，第二拍读取操作数，第三拍修改地址指针即 R_i 内容加 1。ST₁ 与 ST₂ 的操作本可交换，但现在的安排使各种寻址方式在 ST₁ 中的操作相同，有利于简化微命令的逻辑条件。

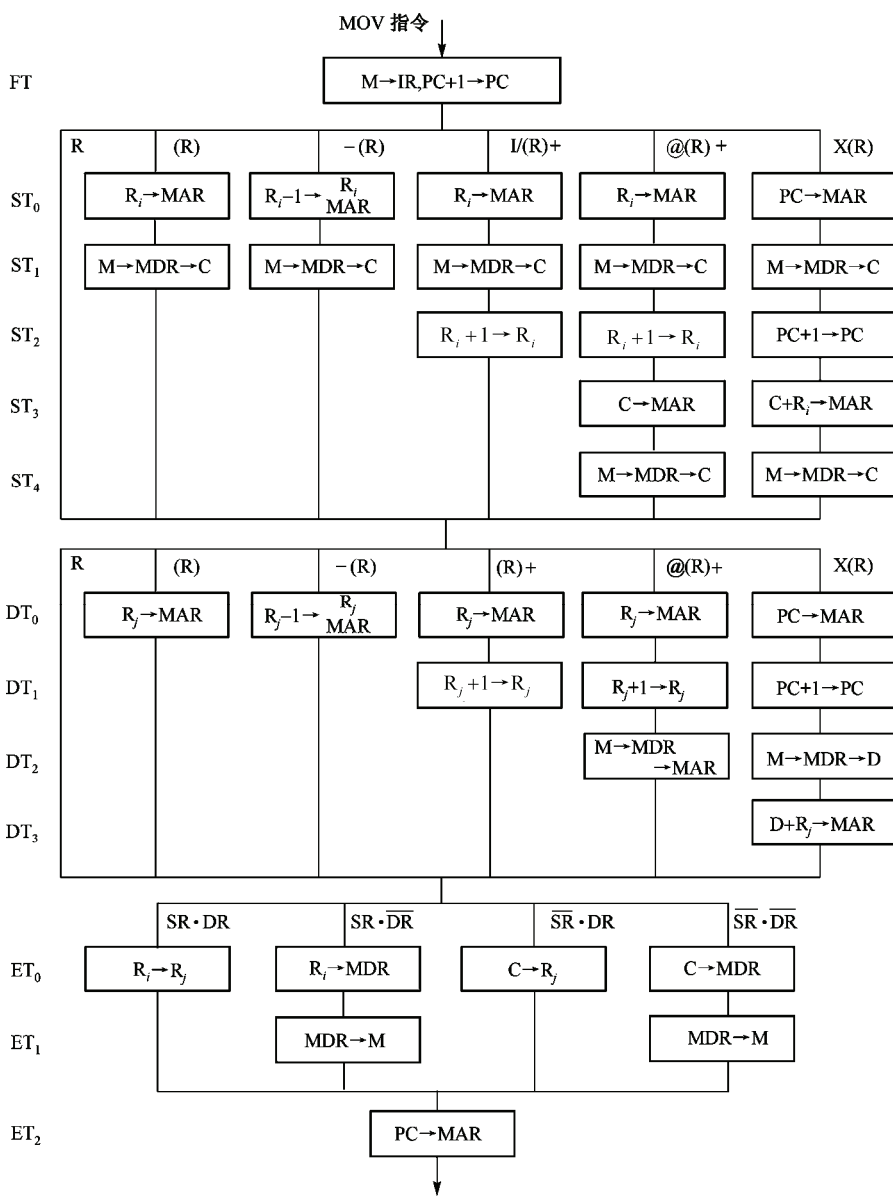


图 3-40 MOV 指令流程图

⑤ $@(R)+$ 型。由于采用双重间址，需两次访存。ST₀取得间址单元地址，ST₁从间址单元中读取操作数地址，ST₂修改指针，ST₃将操作数地址送MAR，ST₄读取操作数。从时间优化角度，可将ST₁与ST₃操作合在ST₁一拍中完成。从保持ST₁操作统一的角度，图中分成两拍。

⑥ $X(R)$ 型。这种寻址方式也需要两次访存，第一次是在PC的指示下读取形式地址，第二次读取操作数。ST₀中将PC内容送MAR，取指后PC已加1，所以此时PC指向紧跟现行指令的下一单元（形式地址存放单元）。ST₁读取形式地址，暂存于C。ST₂再次修改PC指针。在节拍ST₃中完成变址计算，即变址寄存器R_i中的变址量与C中的形式地址相加，形成操作数地址。最后，在节拍ST₄中读取源操作数。

表 3-9 给出了ST₀与ST₁的操作时间表，其余节拍请读者自行拟定。

表 3-9 MOV 指令操作时间表 (ST₀ 和 ST₁)

| | | | |
|-----------------|---|---|--|
| ST ₀ | $R_0 \rightarrow A[\overline{IR_5} \overline{IR_4} \overline{IR_3} \overline{X}]$ $R_1 \rightarrow A[\overline{IR_5} \overline{IR_4} \overline{IR_3} \overline{X}]$ $R_2 \rightarrow A[\overline{IR_5} \overline{IR_4} \overline{IR_3} \overline{X}]$ $R_3 \rightarrow A[\overline{IR_5} \overline{IR_4} \overline{IR_3} \overline{X}]$ $SP \rightarrow A[\overline{IR_5} \overline{IR_4} \overline{IR_3} \overline{X}]$ $PC \rightarrow A[\overline{IR_5} \overline{IR_4} \overline{IR_3} \overline{X}]$ $S_3 S_2 S_1 S_0 M[-(R)]$ $\overline{S_3} \overline{S_2} \overline{S_1} \overline{S_0} \overline{MC_0}[-(R)]$ DM T+1 | P | CPMAR CPR ₀ (同 R ₀ →A 条件) CPR ₁ (同 R ₁ →A 条件) CPR ₂ (同 R ₂ →A 条件) CPR ₃ (同 R ₃ →A 条件) CPSP (同 SP→A 条件) CPT(\overline{P}) |
| | | | |
| ST ₁ | EMAR R SMDR MDR→B $S_3 \overline{S_2} S_1 \overline{S_0} M$ DM $1 \rightarrow ST[(R) \vee \overline{-(R)}]$ $1 \rightarrow DT[(R) \vee \overline{-(R)}] \overline{DR}$ $1 \rightarrow ET[(R) \vee \overline{-(R)}] \overline{DR}$ $T + I[(R) \vee \overline{-(R)}]$ | P | CPC CPST(\overline{P}) CPDT(\overline{P}) CPET(\overline{P}) CPFT(\overline{P}) CPT(\overline{P}) |
| | | | |

由于指令执行流程中包含着若干分支，相应地，在操作时间表中也包含着分支形态。即表中列出的某些微命令是多选一的，对于某一特定指令，只有部分有效，其余的则因逻辑条件不满足而不发。因此，在微命令后面用括号注明其分支逻辑条件，或者根据不同寄存器号，或者由于寻址方式不同。为简化逻辑输入，我们先形成如下一些中间逻辑变量：

X 是变址方式标志，即寻址方式字段代码为 101； \overline{X} 表示寻址方式为非变址型；SR 表示源操作数采用寄存器寻址方式，若不是寄存器寻址方式，则以 \overline{SR} 表示；DR 表示目的地址采用寄存器寻址方式，否则以 \overline{DR} 表示。

表中有 3 种情况进行选择：

① R→A 的选择。这组微命令共 6 个，对某一特定指令只发出其中之一。分支逻辑条件取决于寄存器号及是否变址方式。如果不是变址方式 (\overline{X})，则根据源地址字段中指定寄存器号决定发哪个微命令。如果是变址方式 (X)，或者指定寄存器为 PC，则应发出 PC→A。

② ALU 功能选择。这组微命令共两个，二选一。如果是自减型寄存器间址方式 ($-(R)$)，需执行 A-1 操作，相应的控制命令为 $\overline{S_3} \overline{S_2} \overline{S_1} \overline{S_0} \overline{MC_0}$ 。对于其他几种寻址方式，ALU 只起传送作用，令其输出等于输入 A，相应的控制信号为 $S_3 S_2 S_1 S_0 M$ 。

③ ST₁ 之后的周期状态变化选择情况此时为三选一。如果不是寄存器间址方式或自减型寄存器间址方式 $[(R) \vee \overline{-(R)}]$ (式中 \vee 是逻辑或符号)，则 ST 还需延长，发 $1 \rightarrow ST$ 与 T+1 命令，使 ST 保持为 1，并让节拍计数器 T 计数，由 T₁ 进入 T₂。如果是 (R) 或 $-(R)$ 方式，则结束 ST 并进入新的工作周期状态。若目的段寻址方式是寄存器寻址 (DR)，就应直接进入执行周期，只发 $1 \rightarrow ET$ ；如果是非寄存器寻址 (\overline{DR}) 则将进入目的周期 DT，此时应只发出 $1 \rightarrow DT$ (见表 3-9)。然后，再同时发出 CPST、CPDT、CPET、CPFT 和 CPT。

我们常利用无效操作去简化逻辑条件。如图 3-40 所示，除 $-(R)$ 型外，其他几种寻址方式并不需要将结果送回 R_i，但如果发 CPR_i 将传送结果同时打入 R_i，也不会对 R_i 内容造成任何影响，所以可同时发 CPMAR 与 CPR_i，逻辑条件将得到简化，如表 3-9 中 ST₀ 的脉冲型命令部分。

让我们重复一下在组合逻辑控制器中实现时序状态转换的方法。

① 在每一拍结束时都发 CPT。若 $T+1=0$ ，则 CPT 使 T 复位为 0，结束本工作周期状态。若 $T+1=1$ ，则 T 计数，本工作周期继续，进入新的节拍（时钟周期）。

② 在工作周期结束时，都发 CPFT~CPET，但这组周期状态触发器中只有一个 D 端输入为 1，因而只能维持或进入一种工作周期状态。

(3) 目的周期 DT

与 ST 相似，但对于 MOV 指令，DT 直到取得目的地址为止，不取目的操作数。

(4) 执行周期 ET

执行周期的基本任务是实现操作码要求的传送操作。因而在进入 ET 时，需要考虑操作数是在寄存器中还是在暂存器 C 中？这可以根据 SR 状态区分；结果是送往寄存器还是送往主存？这可以根据 DR 状态区分。由此，按 SR、DR 形成 4 种分支，如图 3-40 所示。

在现行指令周期结束后，从何处取下一条指令？所以在 ET2 中要执行 $PC \rightarrow MAR$ ，即送后继指令地址。这样，在下一个指令周期的 FT 中就可直接读取指令了。当然，也可采取另一种安排方式，即在 FT 中先取指令地址（ $PC \rightarrow MAR$ ），再读取指令。

指令流程图只反映了正常执行程序的情况，实际上在最后一拍还需判别是否响应 DMA 请求与中断请求，即是否发 $1 \rightarrow DMAT$ 或 $1 \rightarrow IT$ 命令。如果都没有，则建立 $1 \rightarrow FT$ ，转入后继指令。

以上是对 MOV 指令的全面分析，存在若干分支选择。如果是一条确定的指令，CPU 将做出唯一的解释与执行。

【例 3-35】 拟出指令“MOV (R_0), (R_1)”的读取和执行流程。

| | |
|--------|-----------------------------------|
| FT | $M \rightarrow IR$ |
| | $PC+1 \rightarrow PC$ |
| ST_0 | $R_1 \rightarrow MAR$ |
| ST_1 | $M \rightarrow MDR \rightarrow C$ |
| DT_0 | $R_0 \rightarrow MAR$ |
| ET_0 | $C \rightarrow MDR$ |
| ET_1 | $MDR \rightarrow M$ |
| ET_2 | $PC \rightarrow MAR$ |

独立地看本指令， ET_0 的操作是多余的。考虑到其他指令的需要，在 DT 中很可能使用 MDR，故在 ST_1 中先将 MDR 的内容送 C 暂存，这样 ET_0 的操作就有必要了。

3. 双操作数指令

双操作数指令共有 5 条：加 ADD、减 SUB、与 AND、或 OR、异或 EOR。它们的指令流程分别如图 3-41 所示。

其中，取指和取源操作数周期与 MOV 指令相同，图中不再细画。目的周期 DT 也与 MOV 指令的 DT 相似，但多一步操作，即访存读取目的操作数，并送入暂存器 D，其余则完全相同，不再赘述。OP 是操作运算符，如 $R_0 \text{ OP } R_1 \rightarrow R_1$ ，若该指令是一条加法指令，则所描述的含义即为 $R_0 + R_1 \rightarrow R_1$ 。

4. 单操作数指令

设计的单操作数指令共有 6 条：求反 (COM)、求补 (NEG)、加 1 (INC)、减 1 (DEC)、左移 (SL)、右移 (SR)，它们的指令流程如图 3-42 所示。

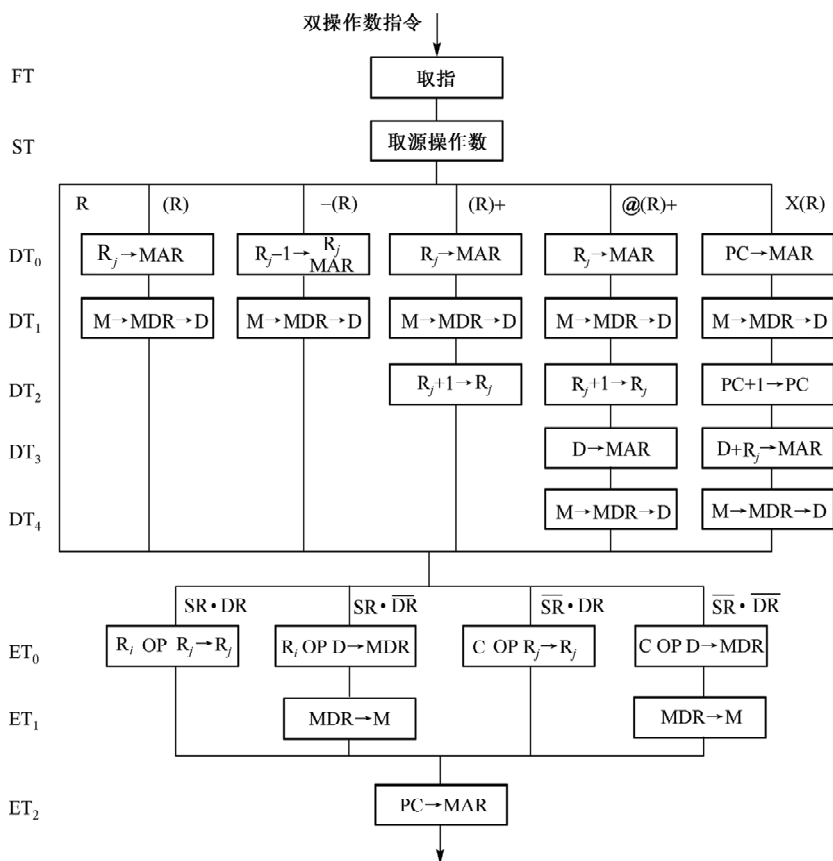


图 3-41 双操作数指令流程图

单操作数指令只有一个操作数，且处理后将会被送回原存储位置保存，因此不需源周期状态 ST，取指后直接进入目的周期 DT。执行周期 ET 中的流程分支也只有两类。其余均与双操作数指令相同。ET₀ 中的具体操作含义取决于操作码 OP 类型，如 OP $R_1 \rightarrow R_1$ ，若该指令是一条求反指令，则所描述的含义是 $\overline{R_1} \rightarrow R_1$ 。

5. 转移指令 JMP/返回指令 RST

JMP 指令的流程如图 3-43 所示。RST 指令被视为 JMP 指令的一种特例。

JMP/RST 指令的主要任务是获得转移地址或返回地址，安排在执行周期 ET 中完成。因此，在 FT 中读得指令并修改 PC 后，直接进入 ET。根据指令规定的转移条件与 PSW 相应位的实际状态，决定是否转移，相应地存在转移成功 (JP) 和转移不成功 (NJP) 两种可能。

(1) 转移不成功 NJP

转移条件不满足，程序将顺序执行。在决定后继指令地址时有以下两种可能的情况：

- ⊙ 转移地址段中未指明 PC (即 $\overline{IR_{11}}\overline{IR_{10}}\overline{IR_9}$)，称为 \overline{PC} 型，则后继指令紧跟着现行转移指令 (在 FT 中修改后的 PC 内容就是后继指令地址)，在 ET 中执行 $PC \rightarrow MAR$ 即可。

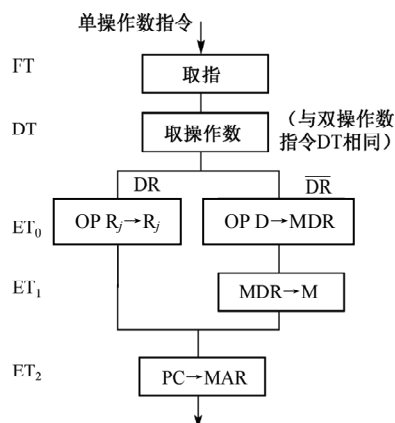


图 3-42 单操作数指令流程图

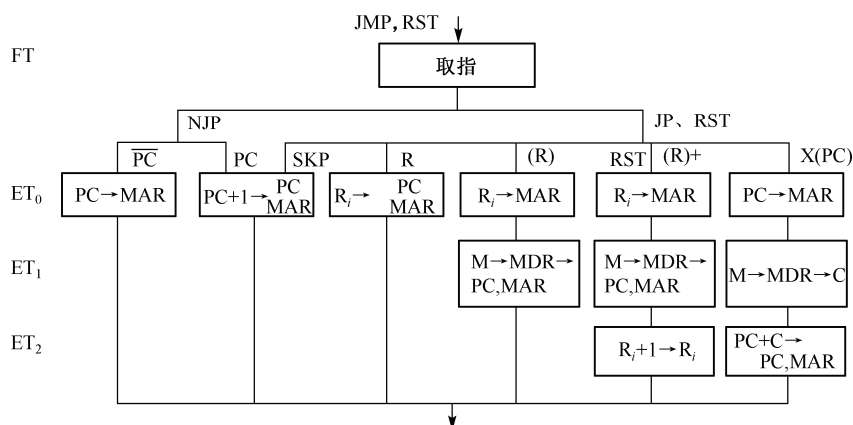


图 3-43 转移指令流程图

◎ 转移地址段中指明 PC ($IR_{11}IR_{10}IR_9$), 称为 PC 型, 则紧跟指令之后的单元已用来存放转移地址, 再下一个存储单元的内容才是后继指令, 所以在 ET 中令 PC 再次加 1。

(2) 转移成功 JP

转移条件满足, 按寻址方式获得转移地址。对于转移指令, 只使用以下 6 种有实用价值的寻址方式:

- ◎ SKP, 跳步执行, 所以在 ET 中 PC 再次加 1。
- ◎ 寄存器寻址 R, 从指定寄存器中读取转移地址。
- ◎ 寄存器间址(R), 从指定寄存器中读取间址单元地址, 再从间址单元中读取转移地址。
- ◎ 自增型寄存器间址(R)+, 比上一种寻址方式增加一步修改指针 R 的操作。
- ◎ 堆栈寻址(SP)+, 用于返回指令 RST, 即从堆栈中读取返回地址, 然后修改指针 SP。
- ◎ 相对寻址 X(PC), 以 PC 内容为基准进行转移地址计算。

6. 转子指令 JSR

在拟定指令系统时, 让转子指令也分为无条件转子和条件转子两种, 以充分利用指令格式的潜力, 使判别与转子相结合。模型机转子指令只采用三种有关的寻址方式, 即寄存器寻址、寄存器间址、自增型寄存器间址, 允许将子程序入口地址存放在寄存器、主存和堆栈中。所指定的寄存器可以是通用寄存器、堆栈指针 SP (以上称为 \overline{PC} 类), 或者程序计数器 PC (称为 PC 类)。返回地址压入堆栈保存, 其流程如图 3-44 所示。

(1) 转子不成功 NJSR。如果转子条件不满足, 则与 JMP 指令中的不转移部分 NJP 相同, 在 ET 中获得后继指令地址。对于 \overline{PC} 类, 则顺序执行; 对于 PC 类, 则跳步执行, 因为紧跟现行指令的存储单元用来存放子程序入口地址, 越过它才是不转子的后继指令。

(2) 转子成功 JSR。如果转子指令采用(R)或(R)+型, 则安排源周期 ST, 从主存中读取转移地址 (子程序入口), 暂存于 C 中。

在 $ET_0 \sim ET_2$ 三拍中, 先保存返回地址, 即修改堆栈指针, 将 PC 内容 (返回地址) 经 MDR 压入堆栈保存。在 ET_3 中再将子程序入口地址送入 PC 和 MAR。

7. 中断周期 IT

以上分析了各类指令的执行流程, 现在讨论 CPU 响应外部请求时的处理过程。图 3-45 给出了中断响应与中断周期的流程, 表明中断周期 IT 是程序切换过程中的一个过渡阶段。

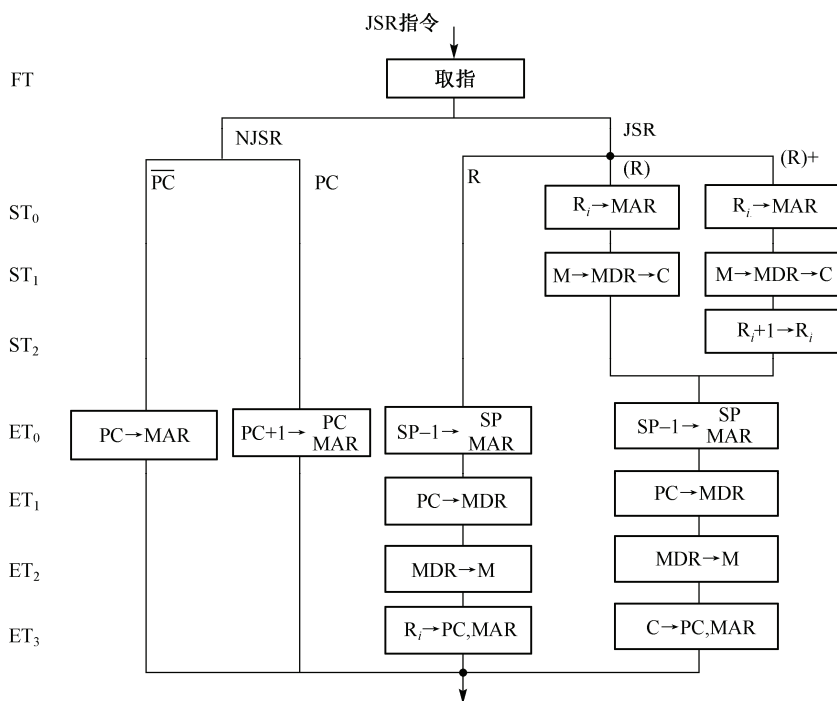


图 3-44 转子指令流程图

(1) 中断响应

若 CPU 在主程序（被中断的程序）第 k 条指令周期中接到中断请求信号 INT，且满足响应中断的条件（如没有 DMA 请求、该中断请求优先级高于现行程序、正在执行的第 k 条指令不是停机指令等），则在该指令周期的最后一拍 ET_i 向请求源发出中断响应信号 INTA，并形成 $1 \rightarrow IT$ ，在周期切换时发出 CPIT。这就使得 CPU 在执行完第 k 条指令后，转入中断周期 IT。

(2) IT 流程

在中断周期中 CPU 暂停执行程序指令，IT 操作依靠纯硬件实现，这类操作常被称为隐指令操作。由于模型机的设计目标是使硬件尽可能简单，因此我们只让 CPU 在 IT 中完成最必须的操作，即为实现程序切换所必须的保存断点、转向服务程序入口。更换程序状态字、保护现场信息这类任务则放在服务程序中完成。

① IT_0 中做两项操作：一是关中断，即让“允许中断”触发器为 0，在响应中断后的过渡阶段暂不响应新的中断请求；二是修改堆栈指针 SP，为保存断点做准备。这两项操作可以同时执行，分别记为： $0 \rightarrow I$ ； $SP-1 \rightarrow SP$ ，MAR。

② IT_1 、 IT_2 中保存断点，即将 PC 内容（后继指令地址）经 MDR 写入主存，也就是将断点压栈保存。

③ IT_3 中将向量地址送 MAR，以便访问中断向量表（有关细节将在第 5 章中讨论）。一种可能的方案是：被批准的中断源通过系统总线向 CPU 提供其中断类型码（如 0 型、1 型……），乘以 4 后转换为中断向量表的地址码，即向量地址。

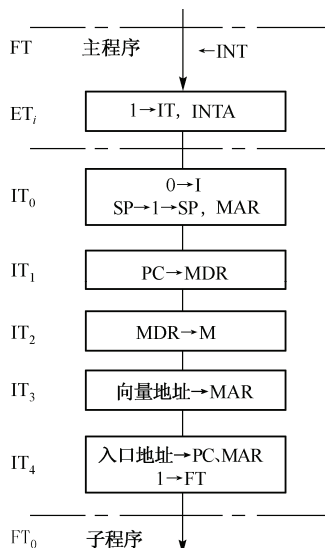


图 3-45 中断周期流程图

④ 在 IT₄ 中访问中断向量表，从中读取对应的中断服务程序入口地址，送入 PC 和 MAR，并形成 1→FT，在 IT 结束时转入取指周期 FT，以便读取中断服务程序。

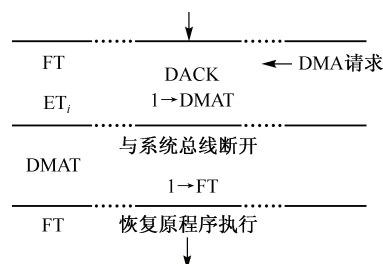


图 3-46 DMA 响应流程

8. DMA 周期

图 3-46 描绘了响应 DMA 请求，进入 DMA 周期实现 DMA 传输的一般过程。

① DMA 响应。在现代的大多数计算机中，允许在一个总线周期结束时响应 DMA 请求，并插入 DMA 周期。相应地，在每个总线周期之后，都要进行一次有无总线请求、是否响应以及总线控制权的切换转移等总线操作。为了减少初学者的学习难度，模型机采用早期计算机在一个指令周期结束时才响应

DMA 请求的做法，这样就能大大简化有关控制逻辑（实际的处理方式更灵活）。

若 CPU 在执行第 k 条指令时接收到 DMA 请求，而第 k 条指令不是停机指令，则在 ET 的最后一拍向外发出批准信号 DACK，并建立 1→DMAT。在周期与节拍切换时，由 CPDMAT 将 DMAT 触发器的输出置为 1，由此进入到 DMA 周期。

② DMAT。在 DMAT 中，CPU 放弃对总线的控制权，即有关输出端呈高阻抗，与系统总线断开。与此同时，DMA 控制器接管系统总线，向总线发出有关地址码与控制信息，实现 DMA 传送。在 DMAT 中，CPU 不做实质性操作，只是空出一个系统总线周期，让主存与外围设备之间实现一次数据直传。但在 DMAT 结束时，将建立 1→FT，以便转入取指周期，恢复原程序的执行。

③ 恢复原程序执行。由于 DMAT 并不影响程序计数器 PC 的内容与有关现场，只是暂停执行程序，所以只要由 DMAT 转入 FT，程序就将恢复执行。

9. 启动与复位

模型机和现在的大多数计算机一样，通过键盘进行启动、复位。有两种情况将导致产生总清信号：上电，即刚加上电源达到正常工作电压时；按复位键，要求程序从头开始。

总清信号将导致 CPU 开始执行监控程序。为此，我们在 0 号单元存放一条转移指令，紧邻的 1 号单元存放转移地址，即监控程序入口地址。

总清信号使 PC = 0、MAR = 0、FT = 1，于是在总清信号结束后，CPU 将从取指周期开始工作。首先从 0 号单元读得 JMP 指令，并从 1 号单元读得监控程序入口地址，然后无条件转向监控程序入口。监控程序使全机初始化，可以接收键盘命令。

3.5.3 微命令的综合与产生

列出所有的操作时间表后，就可以归纳出模型机所需的全部微命令，并综合优化。

1. 微命令逻辑条件的综合化简

首先遍历全部操作时间表，列出全机在各种工作状态下所需的所有微命令，对于可以归并的微命令予以归并优化。在组合逻辑（硬连线逻辑）控制器中，微命令是由组合逻辑电路产生的。具体而言，组合逻辑电路根据输入的操作码、寻址方式和寄存器号等逻辑条件，以及工作周期、节拍序号、定时脉冲等时序条件（见表 3-9）来产生相应的微命令。为了化简，常将这些条件综合为一些中间逻辑变量供使用。组合逻辑电路的输出是所需的微命令，可分为维持一个时钟周期宽的电位型微命令，或是短暂的、靠边沿作用的脉冲型微命令。一台计算机所需的微命令可能有

成百上千或更多，因此组合逻辑控制器中的微命令发生器实际上是一组不规整的组合逻辑电路集合。按这种方式构成的控制器也就被称为组合逻辑控制器。例如，微命令 R/\overline{W} 和 CPR_0 的逻辑式如下：

$$\begin{aligned} R/\overline{W} &= FT_0 + MOV \cdot ST_1 + \cdots \\ CPR_0 &= MOV \cdot ST_0 (\overline{IR}_5 \cdot \overline{IR}_4 \cdot \overline{IR}_3) \overline{X} \cdot P + \cdots \end{aligned}$$

在列出上述微命令的逻辑式时，实际上是归纳了发该微命令时的操作时间表中的公共条件、节拍与周期序号、指令代码（或由此产生的中间逻辑变量）、定时脉冲等。许多微命令将在操作时间表的多处出现，因此微命令的产生条件将呈现“与或”式的逻辑形态。

根据操作时间表列出的微命令逻辑式为初始形态，可以进一步化简。化简的方向一般有两种：一种是提取公共逻辑变量，减少引线，减少元器件数，使成本最低；另一种是使逻辑门级数尽量少，形成命令的时间延迟少，即速度更高。在实际设计中，这两个方向可能有矛盾，应根据需要恰当地选择设计目标。

2. 逻辑实现

传统的方法是直接用门电路实现上述逻辑式，这一组电路就泛称为微命令发生器，它们是控制器的主要实体。

现在广泛采用 PLA 门阵列实现，或者尽量用 PLA 产生大部分微命令，使电路结构得到简化。有的参考书将它称为 PLA 控制器，实际上 PLA 只指所采用的电路形态而已。PLA 的基本结构是译码 - 编码组合。产生微命令的各种逻辑条件作为 PLA 的输入，芯片内部产生相关译码输出，再经编码形成若干微命令输出。这种多输入 - 多输出组合逻辑很适合于构成微命令发生器。

组合逻辑控制器框图见图 3-34，这里不再赘述。

在比较完整地介绍了模型机组合逻辑控制器设计之后，我们将发现组合逻辑控制方式的缺点，并找到相应的改进方向。

① 设计不规整。如前所述，组合逻辑控制方式是用许多门电路产生微命令的，而这些门电路所需的逻辑形态很不规整，因此组合逻辑控制器的核心部分比较烦琐、零乱，设计效率较低，检查调试也比较困难。就其设计方法而言，虽有一定的规律，但对于不同的指令和不同的安排，所构成的微命令形成电路也就不同。改进的方向是将程序技术引入到 CPU 机器的构成级，使设计规整化，即像编制程序那样去编制微命令序列。

② 不易修改或扩展。组合逻辑控制方式的另一缺点是不易修改或扩展指令功能。这是因为设计结果用印制电路板（硬连逻辑）固定下来后，就很难再修改或扩展。而且，在各微命令逻辑式中往往包含着许多条件，其中一些逻辑变量可能是其他微命令所公用的，修改一处就会牵动其他，所以很难改动。机器一旦设计并制作出来，要想修改其操作过程或某些操作的处理方式，或进一步修改或扩充指令系统，基本上是不可能的。这就使一台新设计的机器可能难以达到理想的效果；如果推出一种新的指令系统，原有的机器就不能执行它。改进的方向是将存储逻辑引入 CPU，取代组合逻辑的微命令发生器。也就是将微命令作为数字代码直接存入一个存储器中，只要修改所存储的代码即微命令信息，就可修改有关的功能与执行的方式。

这两种改进方向促使微程序控制思想的出现，但组合逻辑控制方式仍是产生控制命令的一种基本方式；即使采用微程序控制后，仍有部分微命令需由组合逻辑电路产生。组合逻辑控制方式的优点是速度较快，因此目前主要应用于高速计算机的 RISC 处理器和一些巨型计算机中，在一些规模较小的计算机中也有应用。

3.6 微程序控制方式

为克服组合逻辑控制方式存在的缺点，即设计不规整和不易修改，提出了微程序控制方式。本节先介绍有关微程序控制的基本原理和方法，再讨论模型机的微程序设计问题。

采用微程序控制方式产生所需微命令的控制器被称为微程序控制器。所谓微程序控制方式，是指微命令不是由组合逻辑电路产生的，而是由微指令译码产生的。一条机器指令往往分成几步执行，将每步操作所需的若干微命令以代码形式编写在一条微指令中，若干条微指令组成一段微程序，对应一条机器指令。在设计 CPU 时，根据指令系统的需要，事先编制好各段微程序，并将它们存入到一个专用存储器中，称为控制存储器。

图 3-47 表示了微程序控制器的原理结构图。微程序控制器中同样有指令寄存器 IR、程序计数器 PC、程序状态字寄存器 PSW、时序系统等部件。与组合逻辑控制器的最大不同之处是，微命令产生部件的实体发生了变化，不再是一些组合逻辑电路的集合，而是一个控制存储器（Control Memory，CM）和相应的微指令寄存器 μIR ，以及微地址形成电路、微地址寄存器 μAR 等部件。执行指令时，从 CM 中找到相应的微程序段，逐次取出微指令，送入 μIR ，译码后产生所需的微命令，以控制完成各步操作。

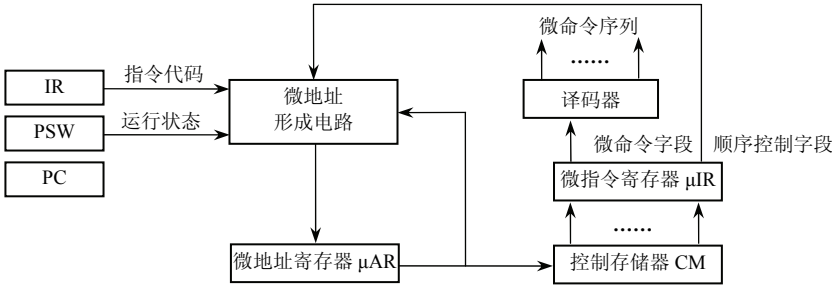


图 3-47 微程序控制器原理框图

CM 通常是一个只读存储器，控制器的外围电路比较规整。指令系统不同，对应的各段微程序也不同，但仅体现为 CM 内容和容量的差异，总体结构差别不大。

3.6.1 微程序控制的基本原理

微程序控制的概念，最早是由英国剑桥大学的威尔克斯在 1951 年提出的，经历种种演变，在只读存储器技术成熟后得到了非常广泛的应用。

1. 基本思想

所谓微程序控制方式，是指微命令不是由组合逻辑电路产生的，而是由微指令经过译码后产生的。机器指令的执行是由一系列微操作来实现的，控制这些微操作的微命令按先后顺序发出。这样就可将一步操作所需的微命令以编码形式编在一条微指令中，并将微指令事先存放在某个专用存储器（即控制存储器）中。当要做这步操作时，再将该条微指令取出，译码后产生所需微命令，控制相应操作。一条机器指令需执行若干步操作，每步操作用一条微指令控制完成，因此需编制若干条微指令。这些微指令组成一段微程序，对应一条机器指令，执行完一段微程序，也就完成了一条机器指令所规定的全部操作。这就是微程序控制的思想，可以概括为以下两点。

① 微命令的产生方式。将控制器所需的微命令以代码（微码）形式编成微指令，存入一个

用 ROM 构成的控制存储器中。在 CPU 执行程序时,从控制存储器中取出微指令,包含了与微命令控制有关的全部操作。与组合逻辑控制方式不同,它由存储逻辑事先存储与提供微命令。这体现了 3.5 节提出的一种改进方向,即将存储逻辑引入到 CPU 中。

② 微程序与机器指令之间的对应关系。将各种机器指令的操作分解为若干微操作序列,每条微指令包含的微命令控制实现一步操作。若干条微指令组成一段微程序,解释执行一条机器指令。针对整个指令系统的需要编制出一套完整的微程序,并固化存储在控制存储器中。这体现了另一种改进方向,利用程序技术去编排指令的解释与执行,也就是将程序技术引入 CPU 的构成级。

2. 逻辑组成

采用微程序控制方式的控制器称为微程序控制器。为了提供机器指令信息,并保证工作程序的连续执行,微程序控制器中也应设置指令寄存器 IR、程序计数器 PC、程序状态字 PSW 等。这与组合逻辑控制器并无区别,因为这是从机器指令和工作程序一级上所需要的部件,对各种控制器类型都是共性的内容。不同之处在于微命令的产生方式,它也在一定程度上影响到时序系统的分级。

在图 3-47 所示的微程序控制器原理框图中,微命令的形成部件不再是一些组合逻辑电路的集合,而是控制存储器及其配套逻辑,它们是微程序控制器的核心部件,包括:

① 控制存储器 CM: 用来存放微程序,控制机器操作的一系列微命令。CM 的每个单元存放一条微指令的代码,用来控制一步操作,通常的长度为几十位。

CM 是一种只读存储器,在制造 CPU 时就写入微程序代码。在 CPU 工作时,CM 只读不写,以确保重要的微程序内容不被破坏。CM 的结构比较规整,不同的指令系统对应的各段微程序一般也不同,但仅体现为 CM 内容和容量的差异,其结构不会受到影响。

② 微指令寄存器 μ IR: 从 CM 中读取的微指令存放其中。微指令分为两部分:一部分是提供微命令的微操作控制字段(也称为微命令字段),占据了微指令的大部分,其代码或直接作为微命令,或分成若干小字段经译码后产生微命令;另一部分称为顺序控制字段(也称为微地址字段),它指明后继微指令地址的形成方式,用于控制微程序的连续执行。 μ IR 将微命令字段送译码器,以便直接译码产生相应的微命令;将微地址字段送微地址形成电路,以便产生后继微指令的地址。

③ 微地址形成电路:根据微程序执行顺序的需要,应当有多种后继微指令地址的形成方式,一般依据下述几种信息中的一部分去形成后继微地址:现行微指令中顺序控制字段(决定形成方式)、现行微指令地址(顺序执行时的基准)、微程序转移时的微地址(由微指令给出其全部或部分高位)、机器指令有关代码(作为微程序分支的依据,如操作码、寻址方式等)、机器运行状态(可作为微程序转移的依据)等。显然,在逻辑实现时采用 PLA 电路是比较理想的。

④ 微地址寄存器 μ AR: 用来暂存微指令在 CM 中的地址,取微指令时,根据 μ AR 的内容定位 CM 的存储单元,与 MAR 的作用很类似。当读出微指令后或完成一个微指令周期操作后,微地址形成电路将后继微地址打入 μ AR,准备取下一条微指令。

3. 工作过程

在微程序控制器中,通过读取微程序和执行它所包含的微命令,去解释执行机器指令。即在执行指令时,从 CM 中定位相应的微程序段,逐次取出微指令,送入 μ IR,译码后产生所需的微命令,以控制完成各步操作。具体过程如下:

① 在微程序中有一条或两三条取机器指令用的微指令,其微命令用来实现机器指令的取指操作,属于微程序中的公用部分。在机器指令周期开始时,先从控制存储器中读取“取指微指令”,

用产生的微命令控制 CPU 访存, 读取机器指令, 送入指令寄存器 IR, 然后修改程序计数器 PC 的内容。

② 根据机器指令中的操作码, 通过微地址形成电路, 找到与该机器指令对应的微程序的入口地址, 即第一条微指令的地址。

③ 逐条取出微指令, 每条微指令提供一个微命令序列, 控制有关的操作。根据机器指令的需要和微指令功能的强弱程度, 一条机器指令所对应的微程序可能不同, 有的可能简单, 有的可能比较复杂, 一般包含分支、循环、嵌入微子程序等程序形态。执行完一条微指令后, 根据微地址形成方法产生后继微地址, 读取下一条微指令。

④ 执行完对应于一条机器指令的一段微程序后, 返回到“取指微指令”, 又开始一条机器指令的取指操作。

3.6.2 微指令编码与微地址形成

如前所述, 控制存储器中每个单元可以存放一条微指令, 每条微指令由微命令字段和微地址字段两部分组成。微命令字段中包含着进行某一步操作所需的微命令信息。如何组织和表示这些微命令呢? 这就涉及微指令的编码方法, 以及微地址字段表明如何形成后继微地址, 使微程序能够连续执行。因此, 微指令的编码方式和微地址的形成方式是微程序设计中要解决的基本问题。

1. 微指令的编码方式

(1) 直接控制法 (不译法)

微指令中控制字段的每一位就是一个微命令, 直接对应于一种微操作。例如, R/\overline{W} 微命令可用 1 位表示, 为 1 是读, 为 0 是写。这种编码方法的优点是简单直观, 缺点是信息效率太低, 若不采取补充措施, 将使微指令变得很长。常见的形态有: 在一条微指令中, 仅部分位采用这种不译法; 或者微指令基本上采用不译法, 但另有补充措施, 以减少总的微指令长度。

(2) 分段直接编译法 (显式编码、单重定义)

将微指令分为若干小字段 (组), 各段独立地通过译码电路定义其编码含义, 一种字段编码对应表示一种微命令 (如 001 表示 CPR_1)。基本的分段原则是: 将同类操作中互斥的微命令归为一组。

提高信息表示效率的基本途径就是采用译码方式定义微命令的含义。但一组译码所产生的信息是互斥的, 即在某一时刻只有一个微命令有效。一次数据通路操作需要同时发出几个微命令, 所以需将一条微指令分成若干段, 分别译码。每组可提供一个有效的微命令, 则一条微指令就可同时提供若干个微命令。每个小组所包含的微命令应当是互斥的, 不会在同一条微指令中同时出现。这是微指令分段的第一个原则。

如果微指令的每个小字段表示同一类型的操作, 则微指令的结构比较清晰, 易于编制微程序, 也易于修改扩充功能。例如, 用一个小字段表示运算器的 A 输入端选择, 命名为 AI 段, 001 表示发微命令 $R_1 \rightarrow A$, 010 表示 $R_2 \rightarrow A \cdots$ 又如, 用一个小字段表示移位功能选择, 00 表示直传, 01 表示左移, 10 表示右移……这是微指令分段的又一个原则。

(3) 分段间接编译法 (隐式编码、多重定义)

如果一个字段的含义不仅取决于本字段的编码, 还兼由其他字段 (或位) 参与解释, 则称为分段间接编译法, 即一种字段编码具有多重定义。这种方法能使微指令编码更为灵活多样化, 可进一步提高信息的表示效率。

【例 3-36】 在微指令中设置解释位或解释字段。

| | | |
|---|----|----|
| 0 | AI | BI |
| 1 | K | |

在大部分微指令中需设置有关数据通路操作的控制字段，如输入选择 AI、BI。但某些微指令中需提供常数 K 或微程序转移地址。可以采取这样的方法：用微指令中的一位来区别，该位称为解释位。若解释位为 0，则相应字段为 AI 及 BI；若解释位为 1，则对应字段为常数字段 K。AI 和 BI 所包含的具体微命令再由这些字段的编码去定义。这种编码方式又称为可解释的字段编译，比分别设置 AI、BI 及 K 字段，能有效地减少微指令长度。

有些 CPU 将微指令分成几大类，如 CPU 操作类、I/O 控制类，由某一字段或状态触发器来区分。例如，某触发器为 0，表示微指令控制 CPU 内部操作，按其需要分段编译产生微命令；若该触发器为 1，表示微指令控制 I/O 操作，另有一套分段定义。这就使一些不同时间使用的功能占据同一段微指令代码位，从而减少微指令长度。有的文献称之为分类编译，常用于大、中型计算机中。

(4) 其他编码方法

前面三种最基本的微指令编码方法在实际机器中常混合使用，即有些字段采用不译法，有些字段为单重定义的直接编译法，有些字段则采用间接编译法。此外，还有一些常用编码方法及相应设计技巧，比如：

① 微指令译码与机器指令译码复合控制。机器指令中常指明与形成操作数有关的寄存器号，这部分可以通过简单的译码形成选择寄存器的控制电位，不必再纳入微指令之中。微指令通过译码给出微命令 $R_i \rightarrow A$ ，由机器指令中的寄存器号进一步确定具体的 R_i 。这是一种减少微指令长度的辅助手段。

② 微地址参与解释微指令代码。某种机器的微指令编码采用直接控制法（即不译法），每位对应于一种微命令，但采取了一种辅助手段使微指令的长度缩短。该机将 26 个只有一条或少数几条微指令才使用的微命令称为局部性微命令，它们共占微指令的一位，其具体含义由该微指令所在的微地址参与解释。比如，从控制存储器 004 单元取出的微指令中，某位是“取指”标志，将从主存中读取的代码送往指令寄存器 IR；而从 011 单元取出的微指令中，该位是“变址”标志，控制实现变址运算。

2. 微地址的形成方式

要使微程序连续地执行下去，这就涉及后继微地址的形成问题，主要考虑顺序执行和转移这两种情况。

(1) 初始微地址的形成

每条机器指令由一段微程序解释执行，其入口就是我们所说的初始微地址。

① 取机器指令。如前所述，设置一、二条“取指微指令”或一小段公用的“取指微程序”，实现取指操作。这小段微程序可从 0#单元或其他特定单元开始。

② 功能转移。取出机器指令后，根据指令代码转换成微程序段的入口地址，称为功能转移。由于机器的指令结构不同，以及采取的实现方法不同，功能转移有以下 3 种方式。

⊙ 一级功能转移。根据指令操作码，一次转移到相应微程序入口。如果操作码的位数与位置是固定的，可让指令操作码作为微地址低位段，这样功能转移很容易实现。例如操作码为 0000，则入口地址为 0...00000；若操作码为 0001，则入口地址为 0...00001。这种方法存

在一个问题：由于机器指令系统中操作码 OP 是一组连续的代码组合，所形成的入口微地址也将是一段连续的区间。如图 3-48 所示，每种操作码只对应一个单元，下一单元已是另一条机器指令对应的入口。所以，这些单元被用来存放转移微指令，通过微指令中的转移地址再无条件转移到真正实现指令功能的微程序段。在早期的微程序设计中常采用这种方式，相应地，在微程序中存在着较多的转移。

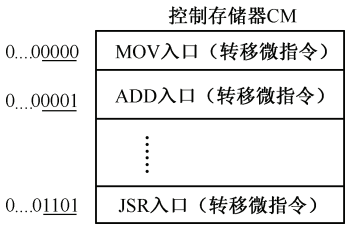


图 3-48 一级功能转移示意图

- ④ 多级功能转移。由于指令功能不仅与操作码有关，还可能与寻址方式等有关，这就可能需要分级转移。例如，先根据操作码实现第一次功能转移后，再根据寻址方式实现第二次转移，以寻找操作数。如果操作码的位数和位置不固定，采用扩展操作码方式，也可能需要分级转移。例如，先根据指令类型的标志位进行分支转移，区分出是哪一大类指令（如双操作数指令，或单操作数指令）。在同一类指令中，操作码位数与位置一般是固定的，因此第二次就可以按操作码转移，区分出是该类指令中的哪一种具体操作。
- ⑤ 采用 PLA 电路实现功能转移。虽然在原理上常需多级转移，才能最后找到与机器指令各段相对应的微程序段，但在 PLA 技术成熟后，可利用 PLA 电路实现快速的一级转移。因为可编程逻辑阵列 PLA 实质上是一种译码 - 编码阵列，具有多个输入和多个输出，可将各种转移依据（操作码、寻址方式等）作为输入代码，对应的输出代码即是相应微程序入口地址。

(2) 后继微地址的形成

找到微程序入口后，可以开始执行微程序。每条微指令执行完毕后，都要根据其顺序控制字段的规定形成后继微地址。后继微地址的形成方法对微程序编制的灵活性影响极大。具体方法很多，可分为以下两种基本类型。

① 增量方式（顺序执行 - 转移方式）

这种方法与工作程序的顺序控制方式相似，即以顺序执行为主，配合各种常规转移方式，故称为增量方式。常见的有：

- ④ 顺序执行，即微地址增量为 1。
- ④ 跳步执行，即微地址增量为 2。
- ④ 无条件转移，即由现行微指令给出转移微地址，或给出全部，或给出低位部分，而高位部分与现行微地址相同。
- ④ 条件转移，即现行微指令的顺序控制字段以编码方式表明转移条件，以及现行微指令的哪些位是转移微地址。
- ④ 微程序转子与返回。常将微程序中的可公用部分编成微子程序，如读取源操作数、读取目的地址等。在微程序中也就相应地有转子与返回等形态。

总之，采用增量方式的优点是直观，与常规工作程序形态相似，容易编制和调试；缺点是难以直接实现多路条件转移，也不容易根据刚形成的运算结果立即转移。为了解释执行各种机器指令，微程序常需多路分支。例如，某指令系统有 16 种操作码，则在功能转移时可能需要实现 16 路分支。因此除了增量方式外，还常采用后面所述的断定方式。

② 断定方式

断定方式是一种直接给定微地址与测试判定微地址相结合的方式。为了实现多路分支，将微

地址的若干低位作为可断定的部分，相应地在微指令的顺序控制段中设置或注明断定条件，即微地址低位段的形成条件。由于分支路数有限，不需将微地址的所有位都作为可断定的，因此只需断定形成有限的低位段，而直接给定高位部分。微指令与微地址的组成如下：

| | | |
|-----|------|------|
| 微指令 | 给定部分 | 断定条件 |
| 微地址 | 高位部分 | 低位部分 |

在微指令中给出两部分信息：直接给定的微地址高位部分和形成低位微地址的方法（即断定条件）。所形成的微地址也由两部分组成：直接给定的高位部分，以及根据断定条件形成的低位部分。所依据的指令代码不同，或依据的运行状态不同，则断定形成的低位微地址不同，分支也就不同。**注意：**断定条件不是低位微地址本身，它只是指明低位微地址的形成条件。

【例 3-37】 假设微地址有 10 位；微指令的断定条件 A 字段有两位，给定部分 D 字段的位数由断定条件确定。

| | | |
|-----|--------|--------|
| 微指令 | 给定部分 D | 断定条件 A |
| 微地址 | 高位部分 | 低位部分 |

约定如下：

- ⊙ A=01——微地址低位段为操作码（若操作码有 4 位，则需给定微地址的高 6 位，而由 4 位操作码指明的微地址低 4 位可实现 16 路分支）。
- ⊙ A=10——微地址低位段为源寻址方式码（若源寻址方式代码为 3 位，则需给定微地址的高 7 位，通过微地址的低 3 位就可实现 8 路分支）。
- ⊙ A=11——微地址低位段为目的寻址方式码（若目的寻址方式码也有 3 位，则需给定微地址的高 7 位，通过微地址的低 3 位就可实现 8 路分支）。

采用断定方式的优点是可以快速实现多路分支转移，适合功能转移的需要；缺点是在编制微程序时，地址安排比较复杂，微程序的执行顺序不直观。在实际的 CPU 中，常混合使用增量方式和断定方式，以使微程序的顺序控制更加灵活方便。

3.6.3 模型机微指令格式

微程序设计的关键在于如何确定微指令格式，它与 CPU 的数据通路结构有很密切的关系。具体地说，就是如何划分微指令的字段，并按所需的微命令分段地分配代码。按照尽可能将同类而互斥的操作命令归为一组这样一个基本原则，沿数据通路的各段操作去划分设置字段。针对图 3-21 模型机数据通路结构的需要，将微命令字段分为如下 3 部分：

- ⊙ 基本数据通路操作的控制字段，其中包含 ALU 的输入选择、ALU 的功能选择、移位选择、内总线输出结果分配。
- ⊙ 访问主存的控制字段，其中包含地址使能、读/写控制。
- ⊙ 辅助操作的控制字段，即将前面两类基本操作未能包括的其他零星操作，如开中断、关中断等单独归为一类，称为辅助操作。出于教学目的，我们有意采用不译法、分段直接编译法、分段间接编译法等多种编码方式。

微指令格式的又一关键，是如何设计微地址字段，它对微程序的编制质量至关重要。究竟需要哪几种后继微地址形成方式，取决于编制微程序的需要。我们采用增量方式实现顺序执行、无条件转移、转微子程序与返回，采用断定方式实现按机器指令操作码、寻址方式与状态字标志位的功能转移。在具体实现方法上还需考虑一个问题，即在断定时如何产生微地址。一种方法是直

接将断定依据作为低位微地址，与给定的高位微地址相拼接，见例 3-37。这种方法比较直观，易于理解，但如图 3-48 所示，将需要安排多次转移，使微程序变得较长。

现在采取另一种方法，将断定依据的代码作为 PROM 的地址输入，从相应单元中读取的内容就是所要形成的后继微地址。这种方法与通过 PLA 产生地址很相似，可使微程序变得比较精练。模型机的微指令格式如图 3-49 所示。

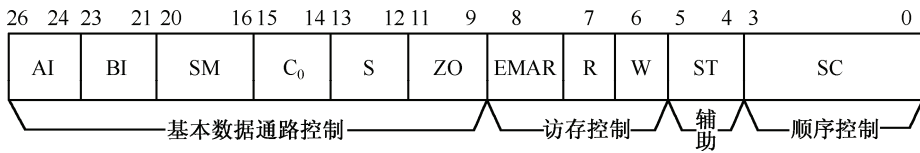


图 3-49 模型机微指令格式

1. 基本数据通路控制字段

- ① AI: ALU 的 A 输入端选择，3 位。
 - ⊙ 000 无输入。
 - ⊙ 001 $R_i \rightarrow A$ (由机器指令中的寄存器号指明 R_i 是哪个寄存器，如 $R_0 \sim R_3$ 、SP、PC)。
 - ⊙ 010 $C \rightarrow A$ 。
 - ⊙ 011 $D \rightarrow A$ 。
 - ⊙ 100 $PC \rightarrow A$ (专用的 $PC \rightarrow A$ 命令，用于取指、变址中对 PC 的选取，而 001 编码中的 $PC \rightarrow A$ 则用于当指定 R_i 为 PC 的寻址)。
 - ② BI: ALU 的 B 输入端选择，3 位。
 - ⊙ 000 无输入。
 - ⊙ 001 $R_i \rightarrow B$ (由机器指令中的寄存器号指明 R_i 是哪个寄存器，如 $R_0 \sim R_3$ 、PSW)。
 - ⊙ 010 $C \rightarrow B$ 。
 - ⊙ 011 $D \rightarrow B$ 。
 - ⊙ 100 $MDR \rightarrow B$ 。
- AI 与 BI 字段都有一些编码组合尚未定义，可用来扩充微命令。
- ③ SM: 即 ALU 功能选择信号 $S_3S_2S_1S_0M$ ，共 5 位，采用直接控制法 (即不译法)。
 - ④ C₀: 初始进位设置，2 位。
 - ⊙ 00 $0 \rightarrow C_0$ 。
 - ⊙ 01 $1 \rightarrow C_0$ 。
 - ⊙ 10 PSW_0 (低位进位标志) $\rightarrow C_0$ 。
 - ⑤ S: 移位器控制，2 位。
 - ⊙ 00 DM (直传)。
 - ⊙ 01 SL (左移)。
 - ⊙ 10 SR (右移)。
 - ⊙ 11 EX (高、低字节交换)。
 - ⑥ ZO: 内总线输出分配，3 位。
 - ⊙ 000 无输出，不发打入脉冲。
 - ⊙ 001 CPR_j (由机器指令中的寄存器号指明 R_j 是哪个寄存器，如 $R_0 \sim R_3$ 、SP、PC、PSW)。
 - ⊙ 010 CPC。
 - ⊙ 011 CPD。

- ⊙ 100 CPIR。
- ⊙ 101 CPMAR。
- ⊙ 110 CPMDR。
- ⊙ 111 CPPC（专用的 CPPC 命令，用于取指、变址中的送入 PC）。

2. 访存操作控制字段

- ⊙ EMAR: 1 位，为 1 时由 MAR 向地址总线提供有效地址，为 0 时 MAR 与地址总线断开。
- ⊙ R: 1 位，为 1 时读主存，同时作为 SMDR。
- ⊙ W: 1 位，为 1 时写入主存，为 0 时 MDR 的输出端与数据总线断开。

以上 3 位采取直接控制法。若 EMAR 为 0，CPU 不访存，但可由 DMA 控制器提供地址。若 R 与 W 均为 0，则主存不工作。

3. 辅助操作控制字段

ST: 2 位。

- ⊙ 00 无操作。
- ⊙ 01 开中断。
- ⊙ 10 关中断。
- ⊙ 11 SIR。

4. 顺序控制字段

SC: 4 位。

- ⊙ 0000 微程序按顺序执行。
- ⊙ 0001 无条件转移，由微指令中第 26~19 位提供 8 位转移微地址。
- ⊙ 0010 按机器指令操作码 OP 进行断定，实现分支转移。
- ⊙ 0011 按 OP 与 DR（即目的寻址是寄存器型还是非寄存器型）断定，分支转移。
- ⊙ 0100 按转移条件是否满足（J 或 NJ）与 PC（指令中指定寄存器是否是 PC）来共同断定，分支转移。
- ⊙ 0101 按源寻址方式断定，分支转移。
- ⊙ 0110 按目的寻址方式断定，分支转移。
- ⊙ 0111 转微子程序，将返回微地址存入一个专设的微地址寄存器中，并由微指令中第 26~19 位提供微子程序入口（约定模型机只允许微程序作单级转子）。
- ⊙ 1000 从微子程序返回，由返回微地址寄存器提供返回地址。

注意：顺序控制字段 SC 本身不是微地址，只是指出形成后继微地址的方法。究竟设置哪些方式，要根据微程序编制的需要。模型机最后确定了 9 种方式，因此 SC 段需要 4 位，还留有若干种编码可供扩充。根据模型机微程序长度，8 位微地址已能满足要求，且留有较大的微程序扩展空间，如编制乘除指令的微程序。根据断定条件与所形成的微地址之间的对应关系，可在微程序编制中产生一个微地址形成表，用 PROM 实现。查找该表，可方便地获取后继微地址。

3.6.4 模型机的微程序设计

为了与组合逻辑控制方式形成对比，将保持模型机的指令系统与格式、数据通路结构以及指令执行流程基本不变，即宏观功能不变，但微操作命令发生器按微程序控制方式形成，相应地，

时序系统也变得简单一些。这样，我们将能更深刻地理解两种控制方式的异同。在讨论了微程序控制的基本原理以及模型机的微指令格式之后，本节主要解决模型机微程序设计中的时序安排和微程序编制等问题。

1. 时序系统

在组合逻辑控制器中，同一指令的分步操作（在不同时间做不同操作）主要依据工作周期状态与时钟周期来划分时间段，所以将工作周期、时钟周期分别作为分段、分步的标志。在微程序控制器中，分步操作依据读取与执行不同的微指令，这些微指令在时序标志上并无区别，只是微指令的代码（微命令信息）不同。因此，微程序控制器的时序系统比较简单且非常规范，不再按阶段设置不同的工作周期（如 FT、ET 等），而采用统一规整的微指令周期。

为了让初学者掌握基本原理，模型机采取最简单的顺序安排方式，即先读取微指令，再开始进行数据通路操作，时序关系如图 3-50 所示。主振荡器输出，经过分频整形，控制脉冲的宽度，

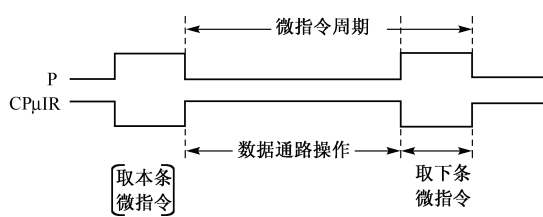


图 3-50 模型机微程序时序图

形成工作脉冲 P，并反相产生打入脉冲 CPμIR。

在 P 的后沿时刻，也就是 CPμIR 的上升沿，将从控制存储器 CM 中读出的微指令送入微指令寄存器 μIR 中。μIR 直接或分段译码产生一组微命令，开始一个微指令周期的数据通路操作。经过一段时间后，已获得稳定的运算结果，在 P 的前沿将结果送入目的地，并将新的微地址

送入微地址寄存器 μAR，又开始读取下一条微指令。P 的宽度足以读取一条微指令，因为 CM 容量较小，工作速度快，与 μAR 及 μIR 的连线很短，所以读取微指令所需的时间是很短的。工作脉冲 P 的相邻两个后沿之间，构成一个微指令周期，也称为一次数据通路周期。在一个微指令周期中，P 的后沿至下一个前沿之间，完成一次数据通路操作；P 的前沿到后沿之间，读取微指令。

为了提高工作速度，实际的 CPU 常采取重叠方式。即将读取微指令安排在两个数据通路操作周期之间，当第一次数据通路操作尚未结束时，就开始读取新的微指令，一旦将新的微指令送入 μIR，就开始新的一次数据通路操作。换句话说，不是以停止 CPU 数据通路操作来为读取微指令安排专门的时间，而是以一种穿插的方式读取微指令，让它与数据通路操作有一些重叠。

如果微程序转移条件与本周期的操作结果有关，则还有些细节问题需要考虑。限于篇幅，这里不再深入讨论。

2. 微程序的编制

编制微程序主要解决编写顺序、实现微程序分支转移等问题。表 3-10 给出了模型机的大部分微程序，留下小部分如“取目的地址”微子程序，可由读者在学习后自行补充完成。表 3-10 中的微程序基本上是按 3.5.2 节的指令流程图编制的，在某些细节上根据微程序特点做了局部调整。

表 3-10 模型机的微程序

| 含义标注 | 微地址 | 操作说明 | 微指令中的各段微命令 |
|------|------|---------|--|
| 取指 | 00 H | M→IR | EMAR, R, SIR, SC = 0000 |
| | 01 H | PC+1→PC | PC→A, S ₃ S ₂ S ₁ S ₀ M̄, C ₀ =1, DM, CPPC, SC = 0000 |
| | 02 H | | 按 OP 分支, SC = 0010 |
| MOV | 03 H | | 转“取源操作数”微子程序入口 4CH, SC = 0111 |
| | 04 H | | 转“取目的地址”微子程序入口 60H, SC = 0111 |
| | 05 H | | 按 OP·DR 分支, SC = 0011 |

| 含义标注 | 微地址 | 操作说明 | 微指令中的各段微命令 |
|-----------------------------|------|---|--|
| MOV· $\overline{\text{DR}}$ | 06 H | $C \rightarrow \text{MDR}$ | $C \rightarrow A, S_3S_2S_1S_0M, DM, \text{CPMDR}, SC = 0000$ |
| | 07 H | $\text{MDR} \rightarrow M$ | EMAR, W, SC = 0000 |
| | 08 H | $PC \rightarrow \text{MAR}$ | $PC \rightarrow A, S_3S_2S_1S_0M, DM, \text{CPMAR}, SC = 0000$ |
| | 09 H | | 转“取指”入口 00H, SC = 0001 |
| MOV·DR | 0A H | $C \rightarrow R_j$ | $C \rightarrow A, S_3S_2S_1S_0M, DM, \text{CPR}_j, SC = 0000$ |
| | 0B H | | 转 08H, SC = 0001 |
| 双操作数 | 0C H | | 转“取源操作数”入口 4CH, SC = 0111 |
| | 0D H | | 转“取目的地址”入口 60H, SC = 0111 |
| | 0E H | $M \rightarrow \text{MDR} \rightarrow D$ | EMAR, R, $\text{MDR} \rightarrow B, S_3\overline{S_2}S_1\overline{S_0}M, DM, \text{CPD}, SC = 0000$ |
| | 0F H | | 按 OP、DR 分支, SC = 0011 |
| ADD· $\overline{\text{DR}}$ | 10 H | $C + D \rightarrow \text{MDR}$ | $C \rightarrow A, D \rightarrow B, S_3\overline{S_2}\overline{S_1}\overline{S_0}\overline{M}, \text{PSW}_0 \rightarrow C_0, DM, \text{CPMDR}, SC = 0000$ |
| | 11 H | | 转 07H, SC = 0001 |
| ADD·DR | 12 H | $C + R_j \rightarrow R_j$ | $C \rightarrow A, R_j \rightarrow B, S_3\overline{S_2}\overline{S_1}\overline{S_0}\overline{M}, \text{PSW}_0 \rightarrow C_0, DM, \text{CPR}_j, SC = 0000$ |
| | 13 H | | 转 08H, SC = 0001 |
| SUB· $\overline{\text{DR}}$ | 14 H | $C - D \rightarrow \text{MDR}$ | $C \rightarrow A, D \rightarrow B, \overline{S_3}S_2S_1\overline{S_0}\overline{M}, 1 \rightarrow C_0, DM, \text{CPMDR}, SC = 0000$ |
| | 15 H | | 转 07H, SC = 0001 |
| SUB·DR | 16 H | $C - R_j \rightarrow R_j$ | $C \rightarrow A, R_j \rightarrow B, \overline{S_3}S_2S_1\overline{S_0}\overline{M}, 1 \rightarrow C_0, DM, \text{CPR}_j, SC = 0000$ |
| | 17 H | | 转 08H, SC = 0001 |
| AND· $\overline{\text{DR}}$ | 18 H | $C \wedge D \rightarrow \text{MDR}$ | $C \rightarrow A, D \rightarrow B, S_3S_2S_1\overline{S_0}M, DM, \text{CPMDR}, SC = 0000$ |
| | 19 H | | 转 07H, SC = 0001 |
| AND·DR | 1A H | $C \wedge R_j \rightarrow R_j$ | $C \rightarrow A, R_j \rightarrow B, S_3S_2S_1\overline{S_0}M, DM, \text{CPR}_j, SC = 0000$ |
| | 1B H | | 转 08H, SC = 0001 |
| OR· $\overline{\text{DR}}$ | 1C H | $C \vee D \rightarrow \text{MDR}$ | $C \rightarrow A, D \rightarrow B, S_3\overline{S_2}\overline{S_1}S_0M, DM, \text{CPMDR}, SC = 0000$ |
| | 1D H | | 转 07H, SC = 0001 |
| OR·DR | 1E H | $C \vee R_j \rightarrow R_j$ | $C \rightarrow A, R_j \rightarrow B, S_3\overline{S_2}\overline{S_1}S_0M, DM, \text{CPR}_j, SC = 0000$ |
| | 1F H | | 转 08H, SC = 0001 |
| EOR· $\overline{\text{DR}}$ | 20 H | $C \oplus D \rightarrow \text{MDR}$ | $C \rightarrow A, D \rightarrow B, S_3\overline{S_2}\overline{S_1}\overline{S_0}M, DM, \text{CPMDR}, SC = 0000$ |
| | 21 H | | 转 07H, SC = 0001 |
| EOR·DR | 22 H | $C \oplus R_j \rightarrow R_j$ | $C \rightarrow A, R_j \rightarrow B, S_3\overline{S_2}\overline{S_1}\overline{S_0}M, DM, \text{CPR}_j, SC = 0000$ |
| | 23 H | | 转 08H, SC = 0001 |
| 单操作数 | 24 H | | 转“取目的地址”入口 60H, SC = 0111 |
| | 25 H | $M \rightarrow \text{MDR} \rightarrow D$ | EMAR, R, $\text{MDR} \rightarrow B, S_3\overline{S_2}S_1\overline{S_0}M, DM, \text{CPD}, SC = 0000$ |
| | 26 H | | 按 OP、DR 分支, SC = 0011 |
| COM· $\overline{\text{DR}}$ | 27 H | $\overline{D} \rightarrow \text{MDR}$ | $D \rightarrow A, \overline{S_3}\overline{S_2}\overline{S_1}\overline{S_0}M, DM, \text{CPMDR}, SC = 0000$ |
| | 28 H | | 转 07H, SC = 0001 |
| COM·DR | 29 H | $\overline{R_j} \rightarrow R_j$ | $R_j \rightarrow A, \overline{S_3}\overline{S_2}\overline{S_1}\overline{S_0}M, DM, \text{CPR}_j, SC = 0000$ |
| | 2A H | | 转 08H, SC = 0001 |
| NEG· $\overline{\text{DR}}$ | 2B H | $\overline{D} + 1 \rightarrow \text{MDR}$ | $D \rightarrow B, \overline{S_3}S_2S_1\overline{S_0}\overline{M}C_0, DM, \text{CPMDR}, SC = 0000$ |
| | 2C H | | 转 07H, SC = 0001 |
| NEG·DR | 2D H | $\overline{R_j} + 1 \rightarrow R_j$ | $R_j \rightarrow B, \overline{S_3}S_2S_1\overline{S_0}\overline{M}C_0, DM, \text{CPR}_j, SC = 0000$ |
| | 2E H | | 转 08H, SC = 0001 |
| INC· $\overline{\text{DR}}$ | 2F H | $D + 1 \rightarrow \text{MDR}$ | $D \rightarrow A, S_3S_2S_1S_0\overline{M}C_0, DM, \text{CPMDR}, SC = 0000$ |
| | 30 H | | 转 07H, SC = 0001 |
| INC·DR | 31 H | $R_j + 1 \rightarrow R_j$ | $R_j \rightarrow A, S_3S_2S_1S_0\overline{M}C_0, DM, \text{CPR}_j, SC = 0000$ |
| | 32 H | | 转 08H, SC = 0001 |
| DEC· $\overline{\text{DR}}$ | 33 H | $D - 1 \rightarrow \text{MDR}$ | $D \rightarrow A, \overline{S_3}\overline{S_2}\overline{S_1}\overline{S_0}\overline{M}, DM, \text{CPMDR}, SC = 0000$ |
| | 34 H | | 转 07H, SC = 0001 |

| 含义标注 | 微地址 | 操作说明 | 微指令中的各段微命令 |
|---------------------|------|-------------------------------------|---|
| DEC·DR | 35 H | $R_j-1 \rightarrow R_j$ | $R_j \rightarrow A, \bar{S}_3\bar{S}_2\bar{S}_1\bar{S}_0\bar{M}, DM, CPR_j, SC = 0000$ |
| | 36 H | | 转 08H, SC = 0001 |
| SL· \overline{DR} | 37 H | $\overline{D} \rightarrow MDR$ | $D \rightarrow A, S_3S_2S_1S_0M, SL, CPMDR, SC = 0000$ |
| | 38 H | | 转 07H, SC = 0001 |
| SL·DR | 39 H | $\overline{R}_j \rightarrow R_j$ | $R_j \rightarrow A, S_3S_2S_1S_0M, SL, CPR_j, SC = 0000$ |
| | 3A H | | 转 08H, SC = 0001 |
| SR· \overline{DR} | 3B H | $\overline{D} \rightarrow MDR$ | $D \rightarrow A, S_3S_2S_1S_0M, SR, CPMDR, SC = 0000$ |
| | 3C H | | 转 07H, SC = 0001 |
| SR·DR | 3D H | $\overline{R}_j \rightarrow R_j$ | $R_j \rightarrow A, S_3S_2S_1S_0M, SR, CPR_j, SC = 0000$ |
| | 3E H | | 转 08H, SC = 0001 |
| JMP 或 JSR | 3F H | | 按 J、PC 分支, SC = 0100 |
| NJ· \overline{PC} | 40 H | | 转 08H, SC = 0001 |
| NJ·PC | 41 H | $PC+1 \rightarrow PC$ | $PC \rightarrow A, S_3S_2S_1S_0\overline{MC}_0, DM, CPPC, SC = 0000$ |
| | 42 H | | 转 08H, SC = 0001 |
| JMP | 43 H | | 转“取源操作数”微子程序入口 4C, SC = 0111 |
| | 44 H | $C \rightarrow PC$ | $C \rightarrow A, S_3S_2S_1S_0M, DM, CPPC, SC = 0000$ |
| | 45 H | | 转 08H, SC = 0001 |
| JSR | 46 H | | 转“压栈”微子程序入口 48H, SC = 0111 |
| | 47 H | | 转 43H, SC = 0001 |
| 压栈 | 48 H | $SP-1 \rightarrow SP$ | $SP \rightarrow A, \bar{S}_3\bar{S}_2\bar{S}_1\bar{S}_0\bar{M}, DM, CPSP, SC = 0000$ |
| | 49 H | $SP \rightarrow MAR$ | $SP \rightarrow A, S_3S_2S_1S_0M, DM, CPMAR, SC = 0000$ |
| | 4A H | $PC \rightarrow MDR$ | $PC \rightarrow A, S_3S_2S_1S_0M, DM, CPMDR, SC = 0000$ |
| | 4B H | $MDR \rightarrow M$ | EMAR, W, 返回, SC = 1000 |
| 取源操作数 | 4C H | | 按源寻址方式分支, SC = 0101 |
| R | 4D H | $R_i \rightarrow C$ | $R_i \rightarrow A, S_3S_2S_1S_0M, DM, CPC, 返回, SC = 1000$ |
| (R) | 4E H | $R_i \rightarrow MAR$ | $R_i \rightarrow A, S_3S_2S_1S_0M, DM, CPMAR, SC = 0000$ |
| | 4F H | $M \rightarrow MDR \rightarrow C$ | EMAR, R, $MDR \rightarrow B, S_3\bar{S}_2S_1\bar{S}_0M, DM, CPC, 返回, SC = 1000$ |
| -(R) | 50 H | $R_i-1 \rightarrow R_i$ | $R_i \rightarrow A, \bar{S}_3\bar{S}_2\bar{S}_1\bar{S}_0\bar{M}, DM, CPR_i, SC = 0000$ |
| | 51 H | | 转 4EH, SC = 0001 |
| (R)+ | 52 H | $R_i \rightarrow MAR$ | $R_i \rightarrow A, S_3S_2S_1S_0M, DM, CPMAR, SC = 0000$ |
| | 53 H | $R_i+1 \rightarrow R_i$ | $R_i \rightarrow A, S_3S_2S_1S_0\overline{MC}_0, DM, CPR_i, SC = 0000$ |
| | 54 H | | 转 4FH, SC = 0001 |
| @ (R)+ | 55 H | $R_i \rightarrow MAR$ | $R_i \rightarrow A, S_3S_2S_1S_0M, DM, CPMAR, SC = 0000$ |
| | 56 H | $R_i+1 \rightarrow R_i$ | $R_i \rightarrow A, S_3S_2S_1S_0\overline{MC}_0, DM, CPR_i, SC = 0000$ |
| | 57 H | $M \rightarrow MDR \rightarrow MAR$ | EMAR, R, $MDR \rightarrow B, S_3\bar{S}_2S_1\bar{S}_0M, DM, CPMAR, SC = 0000$ |
| | 58 H | | 转 4FH, SC = 0001 |
| X(R) | 59 H | $PC \rightarrow MAR$ | $PC \rightarrow A, S_3S_2S_1S_0M, DM, CPMAR, SC = 0000$ |
| | 5A H | $PC+1 \rightarrow PC$ | $PC \rightarrow A, S_3S_2S_1S_0\overline{MC}_0, DM, CPPC, SC = 0000$ |
| | 5B H | $M \rightarrow MDR \rightarrow C$ | EMAR, R, $MDR \rightarrow B, S_3\bar{S}_2S_1\bar{S}_0M, DM, CPC, SC = 0000$ |
| | 5C H | $C + R_i \rightarrow MAR$ | $C \rightarrow A, R_i \rightarrow B, S_3\bar{S}_2S_1\bar{S}_0\bar{M}, DM, CPMAR, SC = 0000$ |
| | 5D H | | 转 4FH, SC = 0001 |
| SKP | 5E H | | 转 41H, SC = 0001 |
| | 5F H | | |
| 取目的地址 | 60 H | | 按目的寻址方式分支, SC = 0110 |
| ... | ... | | ... |

自左向右,表 3-10 的第 1 栏提供有关微程序段含义的标注;第 2 栏是微地址;第 3 栏是该微指令所实现的指令流程操作,为了使指令流程的线索比较清晰,微程序本身的转移性操作标注于第 4 栏;第 4 栏汇集了该微指令所包含的微命令,其中包含电平型与脉冲型,以及顺序控制字段代码,微程序转移、分支则以文字说明。

在表 3-10 中,操作码与寻址方式采用助记符书写。 $DR=1$,表明目的寻址方式是寄存器寻址; $DR=0$,表明目的为非寄存器寻址(即其他几种寻址方式)。这影响到执行操作时,目的数来自 R_j 或暂存器 D,结果送往 R_j 或写入主存。 $J=0$,表明转移条件不成立,记为 NJ; $J=1$,表明转移条件成立,记为 JMP 或 JSR。表中的变量 PC 表明寻址方式是 \overline{PC} 型还是 PC 型,即地址字段中指定的寄存器是否为程序计数器 PC,它影响转移类指令的流程。上述几个中间变量 DR、J、PC 也将分别参与后续微地址的断定,决定微程序的分支转移。

(1) 编制说明

我们采取的编制顺序是先编写取指段;然后按机器指令系统中各类指令的需要,分别编写其对应的微程序;将其中可公用的微子程序,如压栈、取源操作数、取目的地址等,编写并放在后面(高位 ROM 区),在调用处填上有关微子程序入口。

当微程序编写完毕,各段微程序入口已确定,就可以根据需要编制微地址形成表。为了简化微地址形成电路的输入条件,将微地址形成表划分为若干子表,每个子表对应于一种微地址形成方式(一种 SC 代码)。

取指完毕后,微程序按操作码进行第一层分支。MOV 指令转向 03H 微地址单元,双操作数指令(5 种)都转向 0CH 单元,单操作数指令(6 种)都转向 24H 单元,转移与转子指令则转向 3FH 单元。当全部微程序基本上都编好后,这些地址才得以确定。这一层分支是在 $SC=0010$ (按操作码分支)时实现的。

进入 MOV 指令入口 03H 后,通过转微子程序实现取源操作数,暂存于 C 中。返回至 04H,再转微子程序读取目的地址。如果目的地址是寄存器寻址方式,即需将操作数送往某寄存器 R_j ,则立即返回 05H。如果是非寄存器寻址方式,则在微子程序中将目的地址送入 MAR,然后返回至 05H。05H 单元中的微指令实现按 OP 与 DR 分支,引入 OP 作为断定条件之一,是为了与双操作数指令、单操作数指令公用一个微地址形成逻辑,实际上 MOV 指令本身仅需按 DR 分支即可。如果 $DR=0$,则将暂存于 C 中的源操作数写入由 MAR 指示的主存单元,然后传送后继机器指令地址,无条件转向 00H,开始下一个机器指令周期。如果 $DR=1$,则将源操作数由暂存器 C 送目的寄存器 R_j ,然后转 08H,利用 08H、09H 两单元中的微指令进入下一个机器指令周期。

5 条双操作数指令都需要读取源操作数和目的操作数,所以先转向 0CH,执行一段公共的微程序。由于 MOV 指令只需读取目的地址,所以微子程序只编到读得目的地址为止,在 0EH 中再读取目的操作数。对于 $DR=1$ 的指令,0DH 与 0EH 所对应的是多余但无碍的操作,多花费一些时间,但可使微程序精练一些。在 0FH 中按 OP 和 DR 分支,这是第二层分支,直接实现 10 路分支转移(5 种操作码,每种操作码包含两种目的寻址方式)。然后分别编写它们的运算操作部分。若 $DR=0$,则从暂存器 C 与 D 中取操作数,送往 ALU 执行相应运算,结果送 MDR,再写入主存。若 $DR=1$,则从 C 与指定目的寄存器 R_j 中取操作数,运算后送回 R_j 。

单操作数指令与此相似,先转向 24H,读出操作数后再按 OP、DR 分支。第二层分支直接实现 12 路分支转移(6 种操作码,两种目的寻址方式),然后分别编写它们的运算操作部分。

转移类指令先转向 3FH,再按 J(转移条件是否满足)与 PC(指定的寄存器是否是程序计数器 PC)分支。第二层分支直接实现四路分支转移。 $NJ \cdot \overline{PC}$ 为转移条件不成立,且转移地址寻址方

式为非 PC 型，则 PC 内容为后继机器指令地址，转 08H 执行微程序。NJ·PC 为转移条件不成立，但转移地址的寻址为 PC 型，则需 PC+1，才能获得后继机器指令地址。JMP 为转移条件成立，在 43H 中转向微子程序，以读取转移地址。返回 44H，将转移地址送入 PC，然后转 08H。JSR 为转子条件成立，先通过“压栈”微子程序保存返回地址，再通过 43H 和 44H 转向微子程序的入口。

“压栈”微子程序段的作用是保存主程序（工作程序）的返回地址，子程序入口可以采用几种不同的寻址方式，但压栈保存返回地址这一段是相同的，所以编成微子程序共享。

“取源操作数”微子程序是大部分指令都要用的一段微程序。在 4CH 中按源寻址方式直接实现七路分支转移，对应于 7 种寻址方式所需的操作。在 R 型中，将源操作数由 R_i 送入暂存器 C（这一点与组合逻辑控制方式不同），多了一步操作，但简化了微程序的分支。不管源寻址方式如何，在从“取源操作数”微子程序返回时，源操作数都在暂存器 C 中，以后的操作就可统一为从 C 中取数。

表 3-10 末尾只给出了“取目的地址”微子程序的入口为 60H，没有进一步给出具体的微子程序。我们希望作为一项微程序设计的练习，留给读者自己编写补充。此外，对 IT 和 DMAT 等 I/O 操作也暂且留出，请读者自行完善。

（2）微地址形成表

有 9 种方式形成后继微地址，用 SC 字段的 4 位编码表示。其中 4 种是增量方式，剩余 5 种是断定方式。根据 SC 字段可以选择相应的微地址形成表。

- ① SC = 0000，顺序执行微程序。
- ② SC = 0001，无条件转移，微指令高 8 位提供转移微地址。
- ③ SC = 0010，按操作码 OP 分支：

| | |
|-----------|-----|
| MOV | 03H |
| 双操作数 | 0CH |
| 单操作数 | 24H |
| JMP 或 JSR | 3FH |

- ④ SC = 0011，按 OP 与 DR 分支：

| | |
|-----------------------|-----|
| MOV · \overline{DR} | 06H |
| MOV · DR | 0AH |
| ADD · \overline{DR} | 10H |
| ADD · DR | 12H |
| SUB · \overline{DR} | 14H |
| SUB · DR | 16H |
| AND · \overline{DR} | 18H |
| AND · DR | 1AH |
| OR · \overline{DR} | 1CH |
| OR · DR | 1EH |
| EOR · \overline{DR} | 20H |
| EOR · DR | 22H |
| COM · \overline{DR} | 27H |
| COM · DR | 29H |
| NEG · \overline{DR} | 2BH |
| NEG · DR | 2DH |
| INC · \overline{DR} | 2FH |
| INC · DR | 31H |
| DEC · \overline{DR} | 33H |

DEC·DR 35H
SL· $\overline{\text{DR}}$ 37H
SL·DR 39H
SR· $\overline{\text{DR}}$ 3BH
SR·DR 3DH

⑤ SC=0100, 按 J 与 PC 分支:

NJ· $\overline{\text{PC}}$ 40H
NJ·PC 41H
JP 43H
JSR 46H

⑥ SC=0101, 按源寻址方式分支:

R 4DH
(R) 4EH
-(R) 50H
(R)+ 52H
@(R)+ 55H
X(R) 59H
SKP 5EH

⑦ SC=0110, 按目的寻址方式分支:

⋮

⑧ SC=0111, 转微子程序, 微指令高 8 位提供微子程序入口。

⑨ SC=1000, 返回微主程序, 由返回微地址寄存器提供返回地址。

由于需要多个微地址形成表, 在逻辑实现上可以采用数块 PROM, 将形成依据 (查表依据), 如操作码 OP、寻址方式等相应指令代码, 或 DR、J、PC 等中间逻辑变量, 分别作为 PROM 芯片的地址输入, 从相应单元中读取的内容 (8 位) 即所需的后继微地址。例如, MOV 对应的单元中存放微地址 03H, MOV·DR 对应的单元中存放微地址 0AH。SC (顺序控制字段) 代码经译码后, 选取某个 PROM 的输出, 作为在某种后继微地址形成方式下的有效微地址。理论上也可以用一块 PLA 电路形成全部微地址, 但若将各种依据同时输入, 实际的芯片没有那么多输入端, 不如按不同 SC 代码要求, 分别用几块 PROM 芯片实现。

(3) 微指令实例

表 3-10 中的微指令仅标注出有效的微命令。现在以“取机器指令”微程序段为例, 按照模型机的微指令格式, 用代码来表示有关的微指令。

微指令各字段

| 微地址 | AI | BI | SM | C ₀ | S | ZO | EMAR | R | W | ST | SC | 分步操作 |
|-----|-----|-----|-------|----------------|----|-----|------|---|---|----|------|---------|
| 00H | 000 | 000 | 00000 | 00 | 00 | 000 | 1 | 1 | 0 | 11 | 0000 | M→IR |
| 01H | 100 | 000 | 10010 | 01 | 00 | 111 | 0 | 0 | 0 | 00 | 0000 | PC+1→PC |
| 02H | 000 | 000 | 00000 | 00 | 00 | 000 | 0 | 0 | 0 | 00 | 0010 | 按 OP 分支 |

“取机器指令”微程序段包含三条微指令, 分别存放在 0、1、2 号控存单元中。0 号单元中的微指令控制完成 M→IR 操作, 这是一次访存操作, 所以数据通路操作字段的编码为全 0; ST 为 11, 表示用 SIR 命令将读出的指令置入 IR; SC 为 0000, 表示将顺序执行 1 号单元的微指令。1 号单元的微指令控制一次内部数据通路操作, 因此访存操作字段为全 0; AI 选择 PC, BI 选择 0, C₀ 为 1, 故实现 PC+1 操作, 结果送入 PC。2 号单元中的微指令控制按操作码分支, 故 SC 为 0010。

通过前面的分析, 可以看出微程序控制方式具有以下明显的优点。

① 用规整的存储逻辑结构代替了不规整的、复杂的硬连逻辑，使结构得到简化与规整，有利于设计自动化。

② 易于修改和扩展，灵活，通用性强。在数据通路结构不变的前提下，可以通过修改微程序来改变指令的执行方式，通过增加新的微程序来增加新的指令，甚至可用更换一套微程序的办法来更换指令系统。

③ 适于作系列机的控制器。对组合逻辑控制器来说，随着指令系统功能的增加，其价格将迅速增高，控制器逻辑也变得复杂。微程序控制的计算机在同一系列内，功能的增加主要表现为微程序的增加，即控存容量的增加，其他硬件增加不多，所以性能价格比相对较高。

④ 可靠性较高，易于诊断和维护。因为微程序控制器的结构比较规整，易于采用诊断技术。

相对于组合逻辑控制方式，微程序控制方式的主要缺点是速度较慢，因为增加了从控存中读取微指令的时间；由于微程序转移等的结构性问题，也要比数据通路操作本身花费更多的时间。一条微指令的操作比一条机器指令所能定义的操作要简单些，这可能使数据通路结构本来具备的并行操作能力得不到充分发挥，影响执行效率，也就是说，CPU 数据通路结构本来允许并行地做更多的操作，但由于微指令格式的限制，不能在一条微指令中同时定义多个操作，因而不能充分利用数据通路结构的潜力。在功能比较简单的计算机中，由于微程序控制器仍需具备如 ROM、微地址形成部件、微地址寄存器、微指令寄存器等一整套存储逻辑，这时的微程序控制器在性能价格比方面可能不如组合逻辑控制器。上述缺点都与集成电路技术有关。随着集成电路成本的进一步降低，ROM 速度的进一步提高，这些缺点会得到克服。

基于以上分析，微程序控制方式是 CPU 中最广泛采用的控制器组成方式。除了速度要求很高的计算机和功能简单的计算机外，大部分 CPU 都采用微程序控制方式。当然，从目前的发展趋势来看，在采用微程序控制方式的计算机中，也越来越多地通过硬连逻辑方式产生部分微命令，以提高 CPU 的处理速度，如 Pentium 系列机。

3.7 CPU 性能的提升技术

为了提高 CPU 的综合性能，不断有新技术、新工艺被引入到 CPU 的设计和生产过程中，也正是由于这些新技术、新工艺在 CPU 中的应用，才促进了 CPU 结构从简单到复杂，性能从低到高地不断发展。这些被用来提高 CPU 性能的技术主要集中在三方面，第一方面主要是 CPU 的设计技术，第二方面主要是 CPU 的制造工艺新技术，第三方面主要是 CPU 的散热技术。其中，一些对 CPU 性能有着深刻影响的设计技术有诸如流水线、多核、SMT、超线程、CPU 虚拟化等；而制造技术则一般都与大规模集成电路技术相关，如纳米工艺和 3-D 晶体管及光刻技术等。此外，CPU 散热技术方面有半导体制冷片、热管散热、微通道散热和制冷芯片散热技术等。

3.7.1 流水线技术

为提高处理器执行指令的效率，把一条指令的操作过程分成若干个子过程，且每个子过程都在专门功能电路上完成，这样指令的各子过程就能同时被执行，指令的平均执行时间也能大大减少，这种技术称为指令流水线（Instruction Pipeline，IP）技术。

例如，一条指令的执行要经过 3 个阶段：取指、译码、执行，若每个阶段都要花费一个时钟周期，如果不采用流水线技术，那么执行这条指令就需要 3 个时钟周期；如果采用了指令流水线技术，那么当这条指令完成“取指”后进入“译码”时，就可以对下一条指令进行“取指”了，

这样也提高了指令的平均执行效率。

Inel 公司是从 80486 处理器开始采用流水线技术的。下面从时间和空间这个二维层面描述指令流水线的工作原理，以浮点加法指令的运算过程为例，并且忽略了取指等过程，其时 - 空关系如图 3-51 所示。其中，一个浮点加法指令的运算过程分成 4 个子过程，分别是求阶差、对阶、尾数相加和规格化。

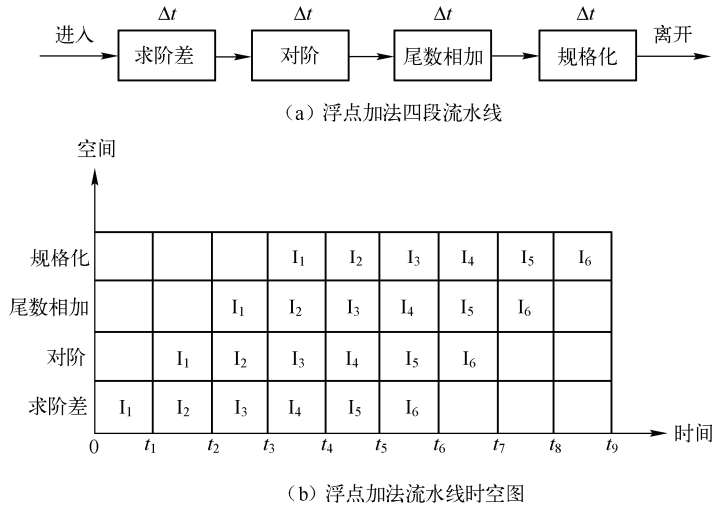


图 3-51 一个浮点加法指令流水线实例

在 $T=0$ 时，指令 I_1 准备进入求阶差。 $T=t_1$ 时， I_1 完成求阶差并准备进入对阶， I_2 准备进入求阶差。 $T=t_2$ 时， I_1 完成对阶并准备进入尾数相加， I_2 完成求阶差并准备进入对阶， I_3 准备进入求阶差。 $T=t_3$ 时， I_1 完成尾数相加并准备进入规格化， I_2 完成对阶并准备进入尾数相加， I_3 完成求阶差并准备进入对阶， I_4 准备进入求阶差。 $T=t_4$ 时， I_1 完成规格化并准备离开流水线， I_2 完成尾数相加并准备进入规格化， I_3 完成对阶并准备进入尾数相加， I_4 完成求阶差并准备进入对阶， I_5 准备进入求阶差。 $T=t_5$ 时， I_2 完成规格化并准备离开流水线， I_3 完成尾数相加并准备进入规格化， I_4 完成对阶并准备进入规格化， I_5 完成求阶差并准备进入对阶， I_6 准备进入求阶差。

以此类推，在 $T=t_9$ 时，最后一条指令 I_6 完成规格化并准备离开流水线。由此可见，采用指令流水线技术后，CPU 在 $9\Delta t$ 时间里能完成 6 条浮点加法指令，平均每条指令的执行时间是 $1.5\Delta t$ ，若不采用流水线技术则需要 $4\Delta t$ ，指令的执行速度加快了约 2.7 倍。注意：指令流水线技术虽然能减少多个指令执行的总时间，但单个指令的执行时间还是保持不变，仍然是 $4\Delta t$ ，多个指令的并行处理只是把总的执行时间减少了而已。

流水线的子过程称为流水线的“段”或“级”，因此流水线“段”的数量也被称为流水线的“流水深度”。

流水线的每个子过程均由专用的硬件功能段实现，各功能段的占用时间也基本相等，如图 3-51(a)中均为 Δt ，通常将 Δt 设置为 1 个时钟周期（即 1 拍）。

流水线需要经过一定的通过时间才能稳定，一般是流水段时间 \times 流水深度，如图 3-51(b)所示的流水线其稳定时间为 $4\Delta t$ 。在某些程序中，同一类指令可能频繁地连续出现，此时采用流水线技术就非常适合，能显著提升 CPU 的运行效率。

1. 流水线的常见种类

作为提升 CPU 性能的一项重要技术，流水线的功能十分繁杂，其种类也很多。如果按照流水线处理的层次级别来归类，大致有三个层次：操作部件级的流水线、指令级的流水线和处理机级的流水线；如果按照流水线能完成的功能的数量来划分，又可以分成单功能流水线和多功能流水线；如果按照流水线内部功能部件的连接方式来划分，又可以分成线性流水线和非线性流水线；如果按照处理的数据对象的形态来划分，又可以分成标量流水线和向量流水线。

(1) 按流水线的处理层级

① 功能部件级：将指令中某个具体的运算，如浮点加法，细分成若干子步骤，各子步骤在功能部件内特定的子功能电路上执行，可同时处理不同指令同一类运算的不同步骤。一般在实现复杂运算时采用，图 3-51 所示的就是一种浮点加法器的部件级流水线。

② 指令级：将一条指令的完整执行过程分为若干子阶段，如取指、译码、执行和回写等，各子阶段在特定的段功能部件上执行，多条指令并行处理。

③ 处理器级：将每个处理任务粗分成若干子任务，各子任务在特定的处理器上执行，同时可并行处理多个任务，这就是处理器级的流水线。在多处理机系统中，经常会使用处理器级的流水线。

(2) 按完成的功能数量

① 单功能流水线 (Unifunction Pipeline)：只能完成一种固定功能的流水线，如乘法或浮点运算流水线等。

例如，Cray-1 计算机有 12 条单功能流水线，我国研制的 YH-1 计算机有 18 条单功能流水线，Pentium 处理机有一条 5 段的整数运算流水线和一条 8 段的浮点运算流水线。采用超流水线体系结构的 Alpha 21064 处理机有三条流水线，其中，整数操作和访问存储器操作为 7 段流水线，浮点运算操作为 10 段流水线。

② 多功能流水线 (Multifunction Pipelining)：各功能部件可以按不同的方式进行连接，以实现多种功能。

多功能流水线的典型代表是 Texas 仪器公司的 ASC 计算机中采用的 8 段流水线。在一台 ASC 处理机内有 4 条相同的流水线，每条流水线通过不同的连接方式可以完成整数加减法运算、整数乘法运算、浮点加法运算、浮点乘法运算，还可以实现逻辑运算、移位操作和数据转换功能等。除了支持标量运算之外，它还支持向量运算，如两个向量的浮点点积运算等。

在处理器中采用多功能流水线的主要优点是能够使流水线中各功能部件的利用率比较高。由于在实际的标量运算程序中，各种类型的运算操作一般是混合在一起进行的，这一点与向量运算操作有很大的不同，因此，在标量计算机的指令执行部件中采用多功能流水线是一种比较合理的选择。与采用多功能流水线不同的另一种方案是设置多条专用的单功能流水线，在许多向量型流水线处理器中通常就是这样设计的。

此外，按照在同一时间内是否能够连接成多种方式以便能同时执行多种功能，又可以进一步把多功能流水线分为静态多功能流水线和动态多功能流水线这两种。

◎ 静态流水线 (Static Pipeline)：指在同一段时间内，多功能流水线的各流水部件只能按一种固定的方式连接，实现一种固定的功能。只有当按照这种连接方式工作的所有任务都流出流水线之后，多功能流水线才能重新进行连接，以实现其他功能。

◎ 动态流水线 (Dynamic Pipeline)：指在同一段时间内，各部件不冲突的前提下，多功能流

流水线中的各流水部件可以按照多种方式连接, 因此可同时执行多种功能。

只有连续出现同一种运算时, 静态流水线的效率才能得到发挥。如果是一连串不同运算操作, 则静态流水线的效率与顺序执行的一样。动态流水线允许两种以上的运算同时执行, 因此效率和部件利用率比静态流水线高, 但它的控制方式更复杂。目前, 大多数处理器中一般都采用静态流水线, 动态流水线很少被使用。

(3) 按处理的数据形态

① 标量流水线: 处理一般形态数据的流水线。

② 向量流水线: 能处理向量数据的流水线, 在大规模的科学计算中经常使用。

(4) 按流水线部件的连接模式

① 线性流水线 (Linear Pipeline): 各流水部件逐个串接起来, 指令从流水线的一端进入, 从另一端输出, 且每个流水部件上只流过一次。

一条线性流水线通常只完成一种固定的功能。在现代计算机系统中, 线性流水线已经被非常广泛地应用于指令执行过程、各种算术运算操作和存储器访问操作等。图 3-51 所示的浮点加法器流水线等都属于线性流水线。

② 非线性流水线 (Nonlinear Pipeline): 各流水部件之间除串行连接以外, 某些部件还可能存在前馈和反馈连接。

(5) 按输入输出顺序的一致性

按照流水线输出端流出的任务与流水线输入端流入的任务的先后顺序是否相同, 流水线可以分为:

① 顺序流水线, 即先进入流水线的指令先从流水线输出, 后进的后出。

② 乱序流水线, 即先进入流水线的指令不一定先从流水线输出, 没有固定的先后顺序, 乱序流水线在有的技术资料上也称为无序流水线、错序流水线或者异步流水线等。

2. 流水线的性能指标

在 CPU 中, 用来衡量流水线综合处理能力的参考指标主要有: 流水线的吞吐率、流水线的使用效率和流水线的加速比。

① 吞吐率。流水线的吞吐率指的是计算机中的流水线在特定时间内可以处理的任务数量或者输出数据结果的多少。吞吐率可以进一步具体化为最大吞吐率和实际吞吐率, 它们主要取决于流水段的处理时间、缓冲寄存器的延迟时间等因素。一般而言, 流水段的处理时间越长, 缓冲寄存器的延迟时间就越大, 则流水线的吞吐率也就越小。

② 加速比。流水线的加速比是指某一程序不采用流水线的实际执行时间与采用了流水线的实际执行时间的比值。由此可见, 加速比的数值越大, 相应流水线的流水安排方式就越好。

③ 使用效率。流水线的使用效率是指流水线中各流水部件的平均利用率。流水线在开始工作时存在建立时间 (即稳定时间), 在结束时存在排空时间, 各流水部件不可能一直都处于工作状态, 总会有一些流水部件在某个时间里处于闲置状态, 因此可以用处于工作状态的部件数量与总的部件数量的比值表征流水线的平均使用效率。

3. 标量和超标量流水线

在标准状态下, 一个处理器一般只含有一条针对同一种功能的流水线, 这就是标量流水线。因此在标量流水线中, 每次只能向流水线输入同一类指令 1 条, 并且每次也只从流水线中输出同一类的流水处理结果 1 个。超标量 (Super Scalar) 流水线则是指在一个处理器中针对同一种功能,

设置有多条并存的流水线，也叫增设冗余流水线。因此，在超标量流水线中，每个时钟周期可向流水线发射同一类的指令 n ($n \geq 2$ ，称为流水线的度) 条，也能从流水线中输出同一类指令的流水处理结果 n 个。这种超标量技术实质是增加同一种功能的流水线的物理数量，以实现同类指令的多条并行处理，从而提高执行速度，本质上还是以空间（流水线数量）换时间（执行速度）。

4. 超级流水线

超级流水线（Super Pipeline）也叫深度流水线，是以增加流水线级数（深度）的方法来缩短机器周期，相同的时间内超级流水线能执行更多的机器指令。

一般而言，CPU 执行一条指令需要经过取指→译码→地址生成→取操作数→执行→写回这几个粗略的阶段。在普通的标量流水线中，每个阶段需要一个标准时钟周期，同时每个阶段的计算结果在周期结束时都要传递到锁存器里供下一个阶段使用。所以，每个标准时钟周期的时间长度应设置为各流水段中耗时最长的流水段时间。若最长的段时间为 S 秒，那么需要的标准时钟周期就是 S ，则其匹配时钟频率就是 $1/S$ 赫兹（Hz）。

因此要提高 CPU 的时钟频率，一种常用的方法就是减小每个原子流水段的时间消耗，这个目标可以通过将每个流水段再进一步细分成更多的子流水段即可实现。经过细分后的每个流水段，单个子段的运算量小了，原子子段的单位耗时 S 也就减少了，与其匹配的时钟周期长度自然也就减小，因而这种方式能提高处理器支持的最大时钟频率。

这种将普通流水线的各流水段进一步细分的技术就是超级流水线技术。在普通标量流水线中，一个标准时钟周期里只有 $m=1$ 条指令进入流水线，而在超级流水线中，一个标准时钟周期里可以有 $m \geq 2$ 条（流水段中的子段数）指令进入到流水线，且进入的指令数 m 等于细分后的子段数，这里的 m 也被称为超级流水线的度。

超级流水线技术实质是以时间换取空间。从原理上看，普通流水线和超级流水线之间并没有本质的区别，只是两者的流水深度不同而已。采用超级流水线技术，虽然从技术上能提高 CPU 的主频，但细分后的每一个子流水段都要使用锁存器锁存中间结果，因此细分为 N 个流水子段并不能让子段时间减少到 S/N ，而是 $S/N+d$ ，其中 d 为锁存时间，这种现象实际上说明细分后反而会额外增加一些不必要的时间消耗；其次，随着流水线深度的增加，指令执行过程中一旦分支预测出现错误，还会导致 CPU 中大量已执行的指令作废，从而花费很大的资源消耗去撤销这些已执行的作废指令，反而会影响 CPU 的处理能力。

5. 超标量超流水线技术

顾名思义，超标量超流水线技术，就是综合使用了超标量技术和超流水线技术。具体而言，就是为同一个功能设置 n 条冗余流水线，然后再对每一条流水线细分成 m 个流水子段。如此以来，超流水的度为 m ，超标量的度为 n ，则超标量超流水的度就为 $m \times n$ 。

6. 与流水线相关的其他技术

（1）超长指令字

超常指令字（Very Long Instruction Word, VLIW）是由美国 Yale 大学教授 Fisher 提出的，有点类似于超标量，是通过一条指令来实现多个操作的并行执行。多个操作之所以要放到同一条指令中，其主要的目的是为了减少内存的访问次数。

通常一条超长指令多达上百位，包含了若干个操作数，每条指令甚至可以执行几种不同的运算。在运行过程中，具体有哪些指令可以并行执行是由编译器来进行优化选择的。通常，VLIW

计算机只有一个控制器，每个周期启动一条长指令，长指令被分为若干字段，每个字段控制与其对应的运算部件。VLIW 计算机对编译器提出了很高的要求，如需要考虑数据的相关性，也要考虑如何避免冲突，还要考虑如何尽可能地利用并行处理技术，甚至要承担指令的运行调度任务。由于编译器接管了本应由硬件承担的工作，所以 VLIW 可以大大简化处理器的硬件结构。

VLIW 对编译器的要求太高，技术难度也太大，实现则更困难，故 VLIW 计算机很少，代表性的是美国加州 Transmeta 公司研制的 X86 指令集 VLIW 系列处理器。

(2) 向量机

普通的计算机一般都只能处理标量，属于标量机。向量机一般都是大型计算机，一般用于军事工业、气象预报及其他大型科学计算领域，这一方面说明了向量机具有高计算性能，另一方面也说明向量机具有较高的研制难度。

国防科技大学研制的“银河”以及“天河”系列超级计算机都属于向量机。普通标量计算机所做的计算，如加减乘除等，只能对一组数据进行操作。向量运算从逻辑上把它等价于对若干同类型标量数据的循环运算，通常是利用流水线技术对大规模数据进行同类操作的批量运算，其运算结果也是一组向量数据。

(3) SIMD 技术

SIMD (Single Instruction Multiple Data) 即单指令多数据。采用 SIMD 架构的 CPU 有多个功能执行部件，但都受控于同一个指令部件。

SIMD 具有较大的性能提升能力。以普通的加法指令为例，单指令单数据 (SISD) 的 CPU 对加法指令译码后，执行部件先访问内存，取得第一个操作数；再次访问内存，取得第二个操作数；最后才能对两个数执行加法操作。在 SIMD 型 CPU 中，指令译码后几个执行部件同时访问内存，一次性获得所有操作数进行运算。

SIMD 具有的这个特点使它特别适合于多媒体应用等领域的数据密集型运算操作。比如，AMD 公司的“3DNow!”技术其实质就是采用了 SIMD，这使它的 K6-2 处理器在音视频和数字娱乐等应用中表现出了优异的性能。

3.7.2 SMT 与超线程

SMT (Simultaneous Multithreading) 即同步多线程，是一种在一个时钟周期内 CPU 能够执行分别来自多个线程的指令的硬件多线程技术（操作系统中是软件多线程）。本质上，CPU 的同步多线程是一种将多 CPU 线程级并行处理转化为单 CPU 指令级并行处理的技术。

SMT 通过复制处理器上的结构状态，让同一个处理器上的多个线程同步执行并共享处理器的执行资源，最大限度实现宽发射、乱序超标量处理，从而提高 CPU 运算部件的利用率，缓和由于数据相关或者 Cache 未命中引起的内存访问延迟。

当没有多个软件线程可用时，SMT 处理器几乎和传统的宽发射超标量处理器一样。采用 SMT 技术最具吸引力的就是只需对处理器内核设计做小规模改动，几乎不用增加额外成本就能显著提升 CPU 性能。SMT 技术可以为高速的运算核心准备更多待处理数据，以减少运算核心的闲置时间。SMT 的优点对于桌面低端系统来说十分具有吸引力，Intel 从 3.06 GHz 的 Northwood 架构 Pentium 4 处理器开始就引入了 SMT 技术。

SMT 仅仅是一个专有名词术语，或者是一类技术思想，也可以看成仅仅是一类 CPU 线程技术的总称。不仅 Intel 在 Pentium 4 中采用，在 Nehalem 架构处理器中也有采用，此外很多其他商用处理器也都在采用 SMT 技术。因此，可以理解成 Nehalem 的 HT 技术和 Pentium 4 的 HT 技术

一样，都属于 SMT 技术思想的一种具体实现。

Intel 对于 SMT 技术的思路是：利用特殊的硬件指令，把两个逻辑内核模拟成两个物理芯片，让单个处理器都能使用线程级并行计算，进而兼容多线程操作系统和软件，减少了 CPU 的闲置时间，提高了 CPU 的运行速度。基于这种思路，Intel 在 2002 年成功开发了基于 SMT 的 HT（Hyper-Threading，即超线程）技术并申请专利，并在 Xeon（至强）处理器上成功应用，随后又将其应用到 Pentium 4、Atom、Itanium 和酷睿 i3/i5/i7 等系列处理器。

采用 HT 技术，就可以在同一时间中让应用程序能够使用芯片的不同部分。虽然单线程芯片每秒钟能够处理成千上万条指令，但是在任一时刻只能对一条指令进行操作。HT 技术则可以使芯片同时进行多线程处理，使芯片性能得到提升。

HT 技术是在一颗 CPU 同时执行多个程序而共同分享一颗 CPU 内的资源，理论上要像两颗 CPU 一样在同一时间执行两个线程，Pentium 4 处理器需要多加入一个 Logical CPU Pointer（逻辑处理单元）。因此新一代的 P4 HT 的晶元面积比以往的 Pentium 4 增大了 5%，但 CPU 的性能可以提升 15%~30%，且其余部件如 ALU、FPU、L2 Cache 则保持不变，这些部件是可以被两个线程共享的。

虽然采用 HT 技术能同时执行两个线程，但它并不像两个真正的 CPU 那样，每个 CPU 都具有独立的资源。当两个线程都同时需要某一个资源时，其中一个要暂时停止，并让出资源，直到这些资源闲置后才能继续。因此超线程的性能并不等于两颗 CPU 的性能。

实际上，某些未进行多线程编译优化的应用程序反而会降低超线程 CPU 的效能。这是因为，超线程 CPU 需要操作系统、主板及应用程序的支持配合，才能充分发挥超线程技术的优点，故支持普通多处理器技术的系统亦未必能充分发挥该技术。

通过 HT 技术，Intel 成为了第一个能在一个实体处理器中实现两个逻辑线程的公司，使其在技术上又领先了 AMD 一步，不得不放弃 SMT 技术的研究，转而将目光集中在双核处理器上。目前的超线程技术一般只能实现两个线程，其未来发展方向，是进一步提升处理器的逻辑线程数量，从而提升处理器性能。

3.7.3 多核技术

多核（Multi-Core）处理器也称为片上多处理器（Chip Multi-Processor，CMP）或者单芯片多处理器，其主要特征是在一个处理器芯片里集成两个甚至多个完整内核（即计算引擎）。多核技术思想早在 1996 年就被斯坦福大学的研究人员提出，但受制于半导体工艺的发展水平，在许多年里多核处理器并没有得到实质性的发展。

1. 多核技术的起源

2005 年之前的主流处理器一般都是单核的。要提高单核处理器的综合性能，通常有两个基本途径：一是提高 CPU 的主频，二是提高每个时钟周期内执行的指令数（Instruction Per Clock，IPC）。虽然处理器微架构的优化可以提高 IPC，采用并行度更高的微架构确实也可以提高 IPC，从而提高处理器的综合性能，但对于同一代微架构的 CPU，通过改良其微架构来提高 IPC 的潜力非常有限，因此在单核处理器时代普遍会采用提高主频的手段来提升 CPU 的综合性能。

然而，通过提高主频来提升处理器的性能，就会使处理器的功耗以指数（主频的 3 次方）的速度急剧上升，带来的散热问题也很难解决。因此，通过无限制地提高主频来提升单核 CPU 性能看来也不可行。就 CPU 的主频而言，目前最大的约为 3~4 GHz，其主频很快就会触及所谓的“频率墙”（Frequency Wall），因此提高主频的路差不多已走到尽头。

在这种情况下，CPU 业界的多数厂商不得不积极寻找另外的途径来提高处理器性能，即多核。由单核处理器增加到双核处理器，即使主频不变，IPC 在理论上也可以被提高 1 倍，因为此时功耗的增加是线性的，故最多也只上升 1 倍。而实际情况是，双核处理器性能达到单核处理器同等性能的时候，双核使用的主频可以更低，功耗也会按主频的 3 次方下降，故双核处理器的起跳主频可以比单核处理器更低，综合性能却更好。

处理器实际性能是处理器在每个时钟周期内所能处理指令数的总量，因此增加一个内核，理论上处理器 IPC 也将增加 1 倍。原因很简单，因为它可以并行地执行指令，故含有几个内核，IPC 的上限就会增加几倍。而在芯片内部多嵌入几个内核的难度远远比加大内核的集成度要低很多。于是，多核技术就能够在不提高设计和制造难度的前提下，用多个低频率内核产生超过单个高频内核的处理效能，特别是服务器产品需要面对大量并行数据时，在多核处理器上分配任务更能够提高工作效率。

2. 多核的种类

从实现技术原理来看，有两种类型的多核，即原生多核和封装多核，而真正意义上的多核指的就是原生多核，最早由 AMD 提出。

原生多核，其特征是各内核之间完全独立，彼此都有自己的前端总线，不会造成冲突，即使多在高负载下，各内核都能保证自己的性能波动不大。因此，原生多核的抗负载能力很强，但其设计和制造都比较复杂。首款原生多核处理器是 AMD 在 2005 年领先于 Intel 发布的 Athlon 64 X2 (K8 架构) 系列处理器。

封装多核，其主要特征是把多个核心直接封装在一起，多个内核之间并不完全独立，且共享第三级缓存和前端总线，因此彼此冲突很大。比如，Intel 早期的 Pentium D8 系列处理器就是把两个单核直接封装在一起而成，与原生多核相比，其综合性能相差还是较大，但其设计和制造成本比较低，因此在初期的时候封装多核比原生多核发展得更快。

3. Intel 与 AMD 的双核技术对比

Intel 和 AMD 的双核处理器都是从多处理器架构发展而来，其基本的变化原理是将两个单独的核心封装在一个 CPU 里面，且每个核心拥有独立的资源，包括处理单元和缓存结构。但是从技术架构上来分析，两者的双核又有很大区别，如图 3-52 所示。

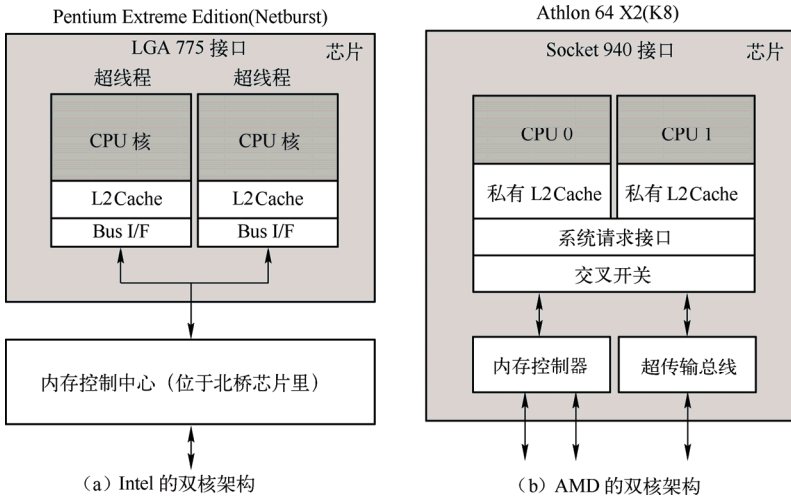


图 3-52 Intel 和 AMD 的双核架构对比

① Intel 的双核（Pentium EE 系列）

Intel 在 2005 年推出的 Pentium EE 双核处理器更像一个双 CPU 平台。Pentium EE 采用 Smithfield 核心，实际上只是将两个 Prescott 核心简单地耦合在一起，每个核心采用独立式缓存(L2)设计，处理器内部两个核心相互隔绝，通过处理器外部位于主板上北桥芯片里的 MCH（Memory Controller Hub，即内存控制中心）仲裁器来负责两个核心之间的任务分配以及缓存数据的同步等协调工作。两个核心共享 FSB（Front Side Bus，前端总线），并依靠 FSB 在两个核心之间传输缓存同步数据。

从体系架构上来看，这种类型是基于独立缓存的松散型双核处理器耦合方案，其优点是技术简单，只需要将两个相同的处理器内核封装在同一块基板上即可；缺点是数据的延迟现象很严重，整体性能并不尽如人意。此外，Pentium EE 还支持超线程（HT），在开启超线程支持后会被操作系统识别为四个逻辑处理器。

② AMD 的双核（Althlon 64 X2）

从设计 K8 时，AMD 就考虑到了未来对多核的支持。AMD 于 2005 年发布的 Althlon 64 X2 双核处理器是把两颗内核集成在一块晶片硅上。为了解决两块核心之间的通信问题，以及对内存和 I/O 带宽的资源争抢，AMD 设计了一个整合在 CPU 芯片内部的 SRQ（System Request Queue，系统请求队列，相当于传统的北桥 MCH），再连接到一个 CS（Crossbar Swith，交叉开关）。如此一来，两个核心之间的协调工作就可以在 CPU 内部完成，不需再借助外部芯片组配合，因而进一步提高了 CPU 的执行效率。同时，SRQ 和 CS 的存在还大大加强了 CPU 的多核扩展性，如要再增加另外的核心，只用把新增的核心连接到 CS 接口上即可，系统架构不需太多改变。

除此之外，AMD 还把内存控制器直接集成在 CPU 内部，这样就可以显著降低 CPU 与主存之间数据交换的时间延迟（不需再像 Intel 架构那样去竞争 FSB），且 AMD 的双核架构中还采用了独有的 HT（HyperTransport，超传输总线标准）。因此，两个核心之间有系统请求队列和交叉开关作为调度器，再加上内置的内存控制器的强力配合，辅助以高带宽的 HT 总线连接，这样就可以提供高效的直接全速通信，不需再从外部总线传输。

由此可见，AMD 将两个内核做一个 Die（晶元）上，通过直连架构连接起来，集成度更高；Intel 则是将放在不同 Die（晶元）上的两个内核封装在一起，因此业界有人将 Intel 的双核称为“双芯”（即封装双核），AMD 的双核才是真正“双核”（原生双核）。

在双核技术上，Intel 明显落后于 AMD，它也认识到这种双核架构的缺点，因此决定放弃 Pentium 4 的 Netburst 处理器架构，全力研究新的 Core（酷睿）架构来对抗 AMD 的 K8 架构，并于 2006 年获得了成功。在 Intel 的 Core 双核架构中，两个核心通过共享 L2 缓存来改善核心之间的通信问题，但在外部仍需共享 FSB。

4. 多核技术的发展

2007 年，AMD 在对 K8 架构进行深度改良以后，继续推出了 K10 架构的 Phenom（羿龙）系列处理器。K10 架构的技术特性主要包括：原生四核设计、引入共享的三级缓存、CPU 独立的供电设计和更灵活的节能机制、HyperTransport 3.0 总线技术和支持 DDR2 内存、SSE 执行单元宽度加倍到 128bit 和支持 AMD-V 虚拟化技术。这些改良措施的确显著提升了处理器的性能，但它仍然无法与英特尔当时的 Core 系列产品相对抗。

2008 年，Intel 发布了基于 Nehalem 微架构的第一代 Core i7 系列处理器（45 nm），第一款桌面型为四核八线程，又在 2009 年发布了四核八线程的 i5 系列处理器，并在 2010 年发布了整合了

GPU (Graphic Processing Unit) 即图形处理器的双核四线程的 i3 系列处理器 (32 nm, Westmere)。Nehalem 架构只是在 Core 微架构基础上增加了 SMT、L3 缓存、TLB、分支预测、IMC、QPI 和支持 DDR3 等技术。Nehalem 与 AMD 的 K10 架构很相似, 但性能更佳、能耗更低。因此, AMD 推出 K10 的改进版本 K10.5 与之抗衡, 其弹性极好, 能衍生成双核、三核、四核以及六核处理器, 能向用户提供丰富的选择空间。

2011 年 3 月, AMD 发布了 Bobcat (山猫) 的第一款融合了 GPU 的 Fusion APU 处理器平台, 分四核和两核规格, 其 CPU 部分为精简的 x86 核心, GPU 则基于 AMD 的 DirectX 11 Radeon 平台, 主要针对超轻薄笔记本, 竞争对手是 Intel Atom。同年 6 月, AMD 代号为 Llano 的主流级 K10.5 架构多核 APU 正式发布, 奠定了图形和异构计算性能方面的优势。

2011 年 4 月, Intel 发布了基于 Sandy Bridge (32 nm) 微架构的四核八线程的第二代 Core i7 系列处理器, 该型处理器实现了 GPU 和 CPU 的完美融合。同年推出了四核四线程的第二代 Core i5 及双核四线程的第二代 i3 系列处理器。此外, Intel 在 2012 年推出了经过改良后的 (Ivy Bridge, 22 nm, 3D - 三栅极晶体管) i7、i5 和 i3 系列处理器。

在 2013 年的台北国际计算机展上, Intel 正式推出基于 Haswell 微架构 (22 nm) 四核八线程的第三代 Core i7, 随后陆续推出了 i5/i3/Xeon 等系列多核处理器, 并计划于 2014 年下半年发布 Haswell 的改进版即 Broadwell (14 nm) 系列处理器。此外, Intel 还计划推出基于 Skylake 微架构 (14 nm) 的多核处理器及其改进版 Skymont (10 nm)。

国产的首款商用多核处理器是 2009 年由中科院计算所研制成功的“龙 3A” (原生四核, 采用 MIPS 64 架构的 RISC 指令集, 65 nm 工艺制程, 主频 1 GHz, 峰值计算能力 16 GFLOPS), 其性能功耗比较高, 已在“曙光” KD-60 服务器上使用。中科院计算所随后在 2012 年推出了“龙芯” 3B-1500 八核处理器, 32 nm 工艺制程, 峰值运算能力可达 192 GFlops, 功耗约为 40~80 W, 还计划推出 16 核的“龙芯-3C”处理器。此外, 国防科大为“天河-1” (2009 年 10 月) 专门研制的“FT-1000” (八核, 六十四线程) 以及为“天河-2” (2013 年 5 月) 研制的“FT-1500” (十六核, 40 nm, 1.8 GHz), 这些都是国产多核处理器的代表之作。

其实早在 2009 年, 处理器已经由双核普遍升级到四核时代, 在斯坦福大学召开的 Hot Chips 大会上, AMD 和 Intel、IBM、富士通等众多芯片制造商展示了其六核、八核等多核服务器专用处理器, 从那开始各处理器产商之间的多核之战就进行得如火如荼。虽然处理器从单核发展出了双核、三核、四核、八核甚至更多核, 但实际上只有双核和四核因其工艺相对简单、成本低、性价比高, 因此这类多核处理器应用最广。原生超过四核的处理器, 如十六核乃至更多核心, 一般都只在服务器甚至巨型计算机中才会使用。

3.8 经典处理器介绍

作为一本计算机专业的基础课教材, 我们选择一种结构简单而规整的模型机来阐明基本原理和设计方法。为了提高 CPU 的性能, 实际的计算机要复杂得多, 如采用多组内部总线、多个运算部件、时间上重叠的流水线技术等。这些技术已成为常规的计算机技术, 除此之外还有一些更加复杂和高级的技术, 如前述内容介绍过的:

- ◎ 向量机, 将某些适于向量化的任务 (如科学计算中的某些任务) 予以向量化并行处理。
- ◎ 超标量方式, 每个时钟周期可执行数条指令。
- ◎ 超长指令字 (VLIW) 技术, 每条指令相当于多条常规指令, 以提高并行度。

在微处理器领域则出现了 RISC 化趋势，即采用精简指令集计算机技术，以提高运行速度。

对于微处理器，CPU 已集成在芯片之内。因此，在构成系统时，我们不必过细地了解其内部具体组成，只需掌握其外特性，即 CPU 芯片有哪些引脚、功能含义、使用方法、时序关系等。从编程角度上说，还需了解其功能结构，即有哪些寄存器，它们的主要功能与使用方法，以及指令系统功能等。事实上，在集成度大大提高之后，我们也越来越难以获得有关计算机的内部细节，只能从功能模型这一层次上去了解各种新出现的典型计算机。

本节选择在微型机、小型机、大型机和巨型机方面几个有代表性的機種，说明实际的 CPU 与模型机相比，可能有哪些变化，以及为提高 CPU 性能所采取的一些措施。

3.8.1 Intel 8086/8088

我们选择 Intel 公司的 8088 作为第一个代表，因为它的内部结构与图 3-21 教学模型机比较相似，但稍复杂一些。在当前应用最广泛的 PC（个人计算机）中，8088 是主流 CPU 系列中最低的一档，尽管已退出历史舞台，但从教学角度比较容易为初学者所理解。

8086 是一种 16 位的 CPU 芯片，内部总线与运算器 16 位，外部系统总线中的数据总线也是 16 位。8088 则是准 16 位 CPU 芯片，内部 16 位，外部 8 位。由于大部分外围设备以字节（8 位）为单位进行数据传送，所以 8088 在连接外围设备方面是比较方便的。在内部组成上 8086 与 8088 基本相同，下面叙述中就只提 8088，其内部组成如图 3-53 所示。

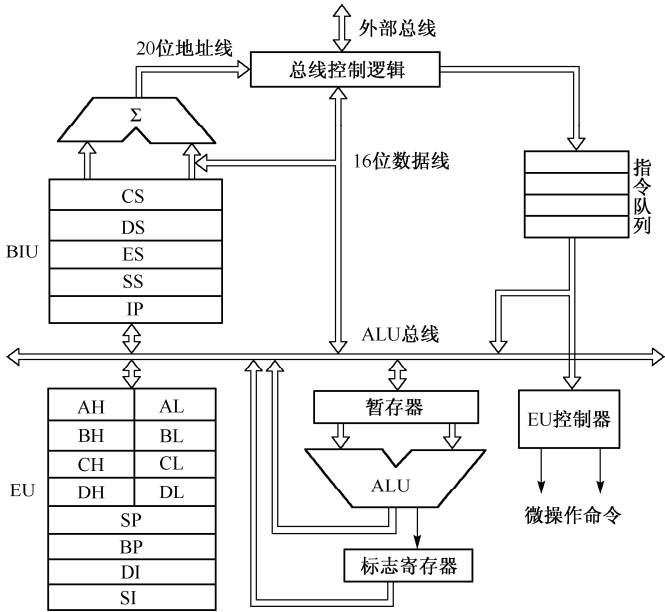


图 3-53 Intel 8086/8088 内部组成框图

8088 内部组成可分为执行单元 (EU)、总线接口单元 (BIU) 两大部分。

执行单元 (EU) 包括一个 16 位算术逻辑运算部件 (ALU)，一组通用寄存器、暂存器、标志 (状态) 寄存器，以及 EU 控制器，大致与图 3-21 所示的模型机组成相当。EU 的任务是从指令队列中取出代码，完成机器指令所规定的功能；当需要访问主存储器或外围设备时，由 EU 向 BIU 发出命令，并提供 16 位的访问地址与需要传送的数据。

总线接口单元 (BIU) 包括一组段寄存器、指令指针 IP (即模型机中的程序计数器 PC)、指令队列 (相当于指令寄存器 IR)、20 位总线地址的形成部件、总线控制逻辑。BIU 的任务是负责

CPU 与主存储器之间，或与外围设备之间的信息传送。

为什么要分成这两部分呢（这也是 8086/8088 与图 3-21 模型机的主要不同之处）？

首先，在模型机中，指令之间的衔接采取串行顺序处理方式，当执行完一条指令之后，才去主存读取下一条指令。在 CPU 执行运算操作时，系统总线与主存储器可能空闲，这使得系统的潜力没有得到充分的发挥。8088 则采取流水处理方式，当 EU 执行指令时，BIU 可从主存中预取后继指令（8088 可预取 4 字节），两部分并行地工作。相应地需要建立一个等待执行的指令队列。这样，BIU 负责从主存中读取指令，交给 EU 去执行（不必等待 EU 执行完毕，BIU 又可预取指令）。虽然这还只是很初级的流水线概念，但已使程序执行速度得到较大的提高。

其次，EU 产生的地址只有 16 位，可访问的存储空间只有 64 KB。为了将微机的主存容量扩充到 1 MB 空间，就需要 20 位地址。怎样将 16 位地址码扩充为 20 位呢？8088 采取这样一种方法：将主存空间划分为若干段，每段 64 KB，相应地，在 BIU 中设置一种段寄存器，其中存放段的首址，又称为段的基址。EU 提供段内的偏移量，即一个存储单元与所在段的基址之间的距离。当 CPU 访存时，一方面指明由哪个段寄存器提供段的基址，同时又给出偏移量，二者错开 4 位相加，由 Σ 产生 20 位地址。

1. 内部组成

下面简要介绍 8088 的内部组成，主要是用户所能看到的部分，或称为功能模型。

(1) 通用寄存器

在 EU 中有 8 个 16 位的通用寄存器，根据它们所担负的基本功能，分为两类。

① 数据寄存器 AX、BX、CX、DX。这 4 个寄存器的每一个既可作为 16 位寄存器，也可以分为两个独立的 8 位寄存器，以便以字节为单位进行处理。因此，它们又分为 AX (AH、AL)、BX (BH、BL)、CX (CH、CL)、DX (DH、DL)，H 表示高 8 位字节，L 表示低 8 位字节。

在大多数情况下，这组寄存器具有一定程度的通用性。在某些指令中，它们被指定担负专门的功能，相应地又有一个专门的名字，即：AX—累加器，BX—基址寄存器，CX—计数寄存器，DX—数据寄存器。

② 地址指针寄存器，包括：SP—堆栈指针，BP—基址指针，SI—源变址寄存器，DI—目的变址寄存器。这 4 个地址指针寄存器可提供前述的偏移量。

(2) 段寄存器

BIU 中有 4 个段寄存器，用来提供段的首址。

⊙ CS—代码段寄存器，代码段中存放指令代码，而 CS 中则存放当前代码段的首址。

⊙ DS—数据段寄存器，数据段中存放着当前程序所需的数据，而 DS 中则存放当前数据段的首址。

⊙ SS—堆栈段寄存器，存放堆栈段首址。

⊙ ES—附加段寄存器，附加段用来保存运算结果或辅助数据，ES 存放该段首址。

(3) 用于控制的寄存器

① 指令指针 IP。IP 的作用与模型机中的程序计数器 PC 相同，在 8088 中叫做指令指针，因为它指示着读取指令的地址。

② 指令队列。指令队列的作用与模型机中的指令寄存器 IR 相似，它是为预取指令而设置的，有 4 字节，又称为指令先行栈。它实际上是 4 个 8 位寄存器。指令队列按先进先出顺序读取，即先进入指令队列的指令，先送往 EU 执行。当指令队列中空出两个字节时，BIU 就从主存中预取指令，存入指令队列。就一条指令而言，是按流水方式读取与执行的，即先由 BIU 从主存中读取

指令，送入队列，再由 EU 从队列中取出，在 EU 中解释执行。从同一时刻看，是重叠处理，即当 EU 在执行一条指令时，BIU 已在读取后面的指令。

③ 标志寄存器。标志寄存器相当于模型机中的程序状态字寄存器 PSW，它有 9 个标志位。其中 6 个是状态标志，即进位位 CF、奇偶位 PF、半进位位 AF（辅助进位位）、零值位 ZF、符号位 SF、溢出位 OF，它们反映指令执行后的结果。另外 3 个是控制标志，即允许中断位 IF、单步标志位 TF、方向标志位 DF。

(4) 暂存器

8088 将暂存器设在 ALU 的输入端。需要运算的操作数先送入暂存器，然后送入 ALU 进行运算处理。如果需要将运算结果写入主存，可先将结果暂存于暂存器，再经总线送至主存储器。

(5) 算术逻辑运算部件 ALU 和地址加法器 Σ

在执行单元 EU 中有一个 ALU。ALU 总线将操作数经暂存器送入 ALU，运算结果也经 ALU 总线送至寄存器或暂存器，并根据运算结果修改相应标志位，如是否有进位、结果是否为 0 等。

在总线接口单元 BIU 中有一个地址加法器 Σ 。段寄存器提供 16 位的段首址，ALU 总线提供 16 位偏移量，二者错 4 位相加，合成为 20 位地址，送往地址总线。

(6) 总线

CPU 内部有一组数据总线，称为 ALU 总线，16 位。寄存器之间、寄存器与 ALU 之间的数据传送，经由这组 ALU 总线。地址加法器 Σ 输出 20 位地址线，ALU 总线向外提供 16 位数据线（输入或输出），再加上一组控制信号线，经过总线控制逻辑，形成 8088 对外部的系统总线。

2. 外部特性

由于 8088 已被集成在一块芯片上，用户更关心的是它的外部特性，即有哪些引脚，以及它们的功能含义和使用方法。8088 采用 40 引脚的双列封装，如图 3-54 所示。

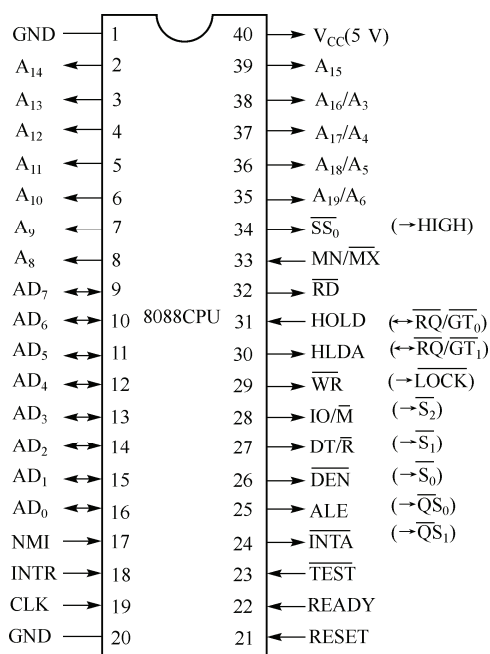


图 3-54 8088 CPU 引脚图

(1) 公共信号线 31 条

① 电源线：+5 V，1 条；地 GND，2 条。

② 地址/数据线（双向）8 条：AD₇~AD₀。

③ 地址线（出）8 条：A₁₅~A₈。

④ 地址/状态线（出）4 条：A₁₉~A₁₆/S₆~S₃。

以上②、③、④三组线采用复用技术。当全部作为地址线时，共 20 位，访存空间 1 MB。

⑤ 控制信号线 8 条：

⊙ 最大/最小模式控制 MN/MX（入）——当用跳线接+5 V 时，系统工作在最小模式。当用跳线接地时，系统工作在最大模式。

⊙ 时钟 CLK（入）——外接时钟信号。

⊙ 复位 RESET（入）。

⊙ 就绪 READY（入）——由主存或 I/O 设备发来的回答信号，表示数据准备好。

⊙ 测试 TEST（入）——在 CPU 执行等待指令后，将测试该引脚状态。若为高，则 CPU 将处于空闲等待状态，系统总线可由其他设备

使用。若为低，表明外部已做好相应工作，CPU 开始执行后面的指令。这种方式可用于 CPU 与外部事件的同步。

- ⊙ 读 RD (出)。
- ⊙ 中断请求 INTR (入)。
- ⊙ 非屏蔽中断请求 NMI (入)。

(2) 最小系统模式信号线 9 条 (当 $\overline{MN}/\overline{MX}=1$ 时)

① 保持请求 HOLD (入)，② 保持响应 HLDA (出)——这两个信号用于 DMA 方式等的总线请求与应答。

- ③ 写 \overline{WR} (出)。
- ④ 数据允许 \overline{DEN} (出)。
- ⑤ 数据发送/接收 DT/\overline{R} (出)。
- ⑥ 地址锁存允许 ALE (出)。
- ⑦ 中断响应 \overline{INTA} 。
- ⑧ I/O、存储器选择 IO/\overline{M} (出)。
- ⑨ 状态信号 \overline{SS}_0 (出)，由 \overline{SS}_0 、 DT/\overline{R} 、 IO/\overline{M} 三位编码指示现行总线周期所处的状态。

(3) 最大系统模式信号线 8 条 (当 $\overline{MN}/\overline{MX}=0$ 时，重新定义 8 条引脚)

- ① 总线请求/应答 RQ/\overline{GT}_0 、 RQ/\overline{GT}_1 (双向)。
- ② 总线优先权锁存 \overline{LOCK} (出)。
- ③ 总线周期状态 S_0 、 S_1 、 S_2 (出)。
- ④ 指令队列状态 QS_0 、 QS_1 (出)。

至此，我们比较详细地介绍了 8088 的内部组成与外部特性，作为从教学模型机到实际 CPU 的过渡，后面将要介绍的几种 CPU 采用了更复杂的技术，还需要更多的知识才能理解。本书只能简略介绍，让读者初步了解当前实用的主流 CPU 的大致结构，进一步的了解留待后续课程。

3.8.2 Intel 80386/80486

1985 年，Intel 公司推出了一种 32 位的微处理器芯片 80386，标志着微型计算机进入 32 位时代。1989 年，Intel 公司宣布了 80486，它相当于一个增强型的 80386、一个浮点处理部件、一个高速缓存 (Cache) 的集成，基本上沿用了 80386 的体系结构。到 1993 年为止，80486 是 32 位微机中最主要的 CPU 芯片，其主要性能如下：

- ⊙ 32 位地址，可直接寻址的物理存储空间为 4 GB (4×10^9 字节)。
- ⊙ 具有存储管理部件，使虚拟存储空间 (逻辑地址空间) 可达 64 TB (64×10^{12} 字节)。
- ⊙ 高速缓存容量 8 KB。
- ⊙ 字长 32 位，系统总线的数据通路宽度 32 位。
- ⊙ 采用 5 级流水线。
- ⊙ 运算速度约为 20 MIPS。

80486 的内部结构如图 3-55 所示，与 80386 结构相似，增加了浮点部件与高速缓存，内部控制方式也有一些改进。

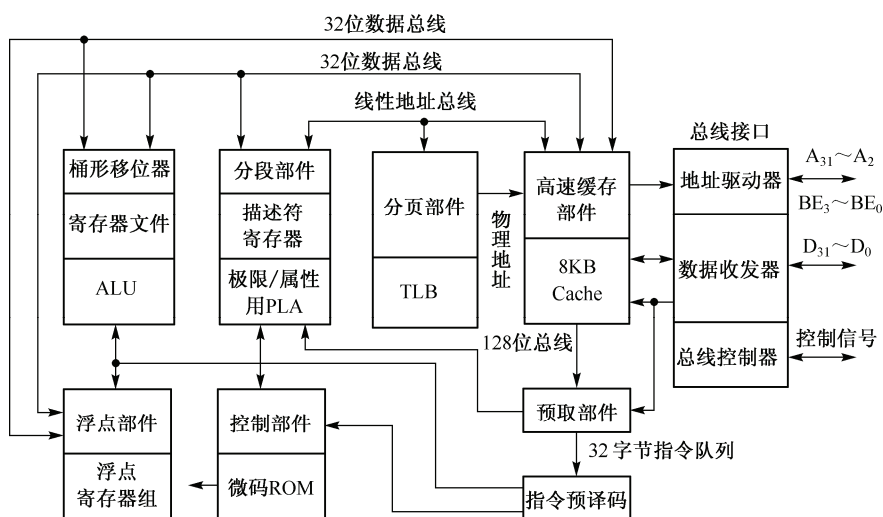


图 3-55 80486 内部结构框图

（1）通用寄存器

与 8086/8088 相比，80386/80486 将 8 个通用寄存器扩充为 32 位，分别命名为 EAX、EBX、ECX、EDX、ESI、EDI、EBP、ESP。它们的低 16 位可以单独访问，又可进一步分为高位字节与低位字节单独访问，命名与 8086/8088 相同，即 AX（AH、AL）、BX（BH、BL）、CX（CH、CL）、DX（DH、DL），因此在目标代码级上与 8086/8088 相兼容。这 8 个通用寄存器位于寄存器组中。

（2）段寄存器与描述符寄存器

80386/80486 设置了 6 个 16 位段寄存器，又称为选择器，分别是：代码段寄存器 CS，数据段寄存器 DS，堆栈段寄存器 SS，附加段寄存器 ES、FS、GS。

为了描述各段的基址、限制和其他属性，为每个段定义一个描述符，这 6 个描述符寄存器位于分段部件中。这些段描述符构成了全局描述符表（GDT）和局部描述符表（LDT）。段寄存器（选择器）中的内容用来说明该段的描述符在哪个表中，序号是多少及特权级的高低。因此，段寄存器是用户可见的，而段描述符寄存器对用户是透明的。

（3）用于控制与调试的一组寄存器

① 指令指针 EIP，32 位。

② 标志寄存器 EFLAGS，32 位，包含状态、控制、系统三种标志，它们是 8086/8088、80286 标志位的进一步扩充，分别有：进位标志位 CF、奇偶标志位 PF、辅助标志位 AF、零标志位 ZF、符号位 SF、自陷位 TF、中断允许标志位 IF、方向标志位 DF、溢出标志位 OF、I/O 特权级标志 IOPL、嵌套任务标志 NT、恢复标志 RF、虚拟 8086 模式标志 VM、对齐检查位 AC。还有一些位为 0，空着未用。

③ 4 个 32 位控制寄存器 CR0、CR1、CR2、CR3，用来存放全局特性的机器状态，其中 CR1 暂时未用，留待新产品扩展。

⊙ CR0：包含有分页允许位 PG、高速缓存不允许位 CD、不透写位 NW、对齐标志位 AM、写保护位 WP、数字错误位 NE、协处理器类型位 ET、任务切换位 TS、仿真协处理器位 EM、监视协处理器位 MP、保护允许位 PE。

⊙ CR2：存放发生页故障的线性地址。

⊙ CR3：存放页目录基址。

④ 系统地址寄存器，包括：全局描述符表寄存器 GDTR，局部描述符表寄存器 LDTR，中断描述符表寄存器 IDTR，任务状态段寄存器 TR。将在保护模式中常用的数据结构基址、限制和其他属性等保存起来，供快速查用。

⑤ 调试寄存器 DR₀~DR₇，共 8 个，32 位可编程寄存器，用来支持调试功能（如存放断点地址），说明断点性质、断点发生条件和类型等。

⑥ 测试寄存器 TR₀~TR₇：TR₀~TR₂ 尚未定义，TR₃~TR₅ 用于控制对 Cache 的测试，TR₆、TR₇ 用于控制对转换后备缓冲器 TLB 的测试。

（4）总线接口部件 BIU

BIU 包含地址驱动器、数据总线收发器、总线控制器，提供 CPU 与系统总线之间的高速接口。

（5）指令部件

指令部件包含指令预取部件、32 字节的指令队列、指令预译码、产生微命令的控制部件。80386 采取微程序控制方式，指令预译码实现由机器指令到微指令的转换，由于采取重叠方式，可以节省时间。80486 采取硬连逻辑控制与微程序控制方式相结合，并引用了部分 RISC 技术，以缩短指令执行时间。在 80486 中，一些基本指令如加减、取数、存数等，由硬连逻辑执行，可在一个时钟周期内完成；一些复杂指令则仍由微程序实现。这种方式使 80486 的运算速度大大提高。

（6）执行部件

执行部件包含前述 8 个 32 位通用寄存器、1 个 64 位的桶形移位器、1 个包含乘除功能的 ALU。通用寄存器又称为寄存器文件，既可以用于数据操作，又可用于地址计算。桶形移位器可在一个时钟周期内实现任意位数的移位。

（7）浮点处理部件

与 80386 比，80486 芯片内增加了一个浮点处理部件 FPU，直接具备浮点处理能力。该部件由指令接口、数据接口、运算控制单元、浮点寄存器、浮点运算器等部分组成。

（8）高速缓存部件

80486 在片内增加一个 8 KB 的高速缓存与相应的控制部件，可由指令与数据共用，约有 92% 的命中率。这是 80486 整体性能大幅度提高的又一原因。在高速缓存与指令预取部件间有 128 位宽的代码总线，这就防止了一般 CPU 常有的总线瓶颈问题。高速缓存与两个运算部件之间有两组 32 位数据总线，也可合并为一组 64 位数据总线，以增强内部数据的传输能力。

（9）存储管理部件 MMU

存储管理是操作系统的重要功能之一。在早期的计算机中，靠操作系统软件来实现；而 80386/80486 在片内集成了存储管理部件，使存储管理操作得以快速实现。

在 80386/80486 中，将存储器按段来组织，以适应用户程序的逻辑结构。段的大小可变，最大可达到 4 GB。针对主存物理空间的组织，又可将存储器划分为页，每页大小一定，为 4 KB。这样，80386/80486 可支持段式存储管理、页式存储管理，或段页式存储管理等方式。在后一种方式中，将用户程序与数据分段，段中又分页。相应地，存储管理部件包括分段部件、分页部件。

段式管理机构将段基址加上有效地址，形成线性地址，即：线性地址=段基址+有效地址。

页式管理机构通过页变换，再将线性地址转换成主存的物理地址。

为了支持多处理机系统的构成，80486 增设了两条专用指令及一些功能和信号。

80486 的引脚信号如图 3-56 所示。限于篇幅，不再解释它们的含义。

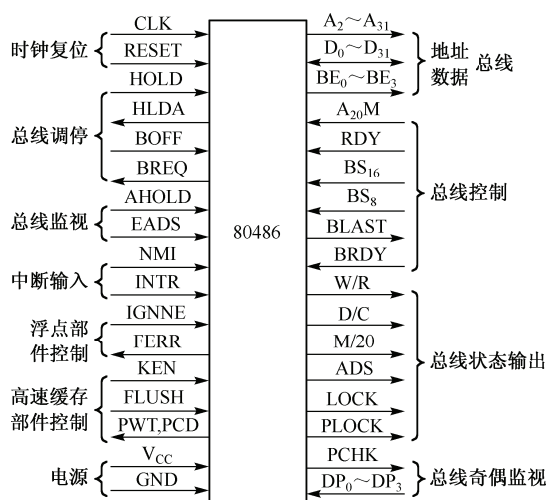


图 3-56 80486 的引脚信号

3.8.3 Pentium 微处理器

Pentium 微处理器是 Intel 80x86 系列微处理器的第五代产品，其性能比它的前一代产品又有较大幅度的提高，但仍保持与 8086、80286、80386、80486 兼容。Pentium 微处理器芯片规模比 80486 芯片大大提高，除了基本的 CPU 电路外，还集成了 16 KB 的高速缓存和浮点协处理器。芯片引脚增加到 270 多个，其中外部数据总线为 64 位，在一个总线周期内，数据传输量比 80486 增加了 1 倍；地址总线为 36 位，可寻址的物理地址空间高达 64 GB。Pentium 微处理器具有比 80486 更快的运算速度和更高的性能。微处理器的工作时钟频率可达 66~200 MHz。在 66 MHz 频率下，指令平均执行速度为 112 MIPS，与相同工作频率下的 80486 相比，整数运算性能提高 1 倍，浮点运算性能提高近 4 倍。常用的整数运算指令与浮点运算指令采用硬件电路实现，不再使用微码解释执行，使指令的执行速度进一步加快。Pentium 微处理器还是第一个实现系统管理方式的高性能微处理器，它能很好地实现微机系统的能耗和安全管理。

Pentium 微处理器之所以有如此高的性能，在于该微处理器体系结构采用了一系列新的设计技术，如双执行部件、超标量体系结构、集成浮点部件、64 位数据总线、指令动态转移预测、回写数据高速缓存、错误检测与报告等。Pentium 微处理器的功能结构框图示于图 3-57 中。

(1) 超标量体系结构

Pentium 微处理器具有三条指令执行流水线：两条独立的整数指令流水线（分别称为 U 流水线与 V 流水线）和一条浮点指令流水线。两条整数指令流水线都拥有它们独立的算术逻辑运算部件、地址生成逻辑和高速数据缓存接口。每个时钟周期可以同时执行两条指令，因而相对同一频率下工作的 80486 来说，其性能几乎提高了 1 倍。我们把这种能一次同时执行多条指令的处理器结构称为超标量体系结构。

Pentium 微处理器的整数指令流水线与 80486 相似，也具有指令预取、指令译码、生成地址和取操作数、指令执行、写操作数等五级。每一级处理需要一个时钟周期。当流水线装满时，指令流水线以每个时钟周期一条指令的速率执行。

(2) 浮点指令流水线与浮点指令部件

浮点指令流水线具有 8 级，实际上是 U 流水线的扩充。U 流水线的前四级用来准备一条浮点指令，浮点部件中的后四级执行特定的浮点运算操作并报告执行错误。此外在浮点部件中，对常

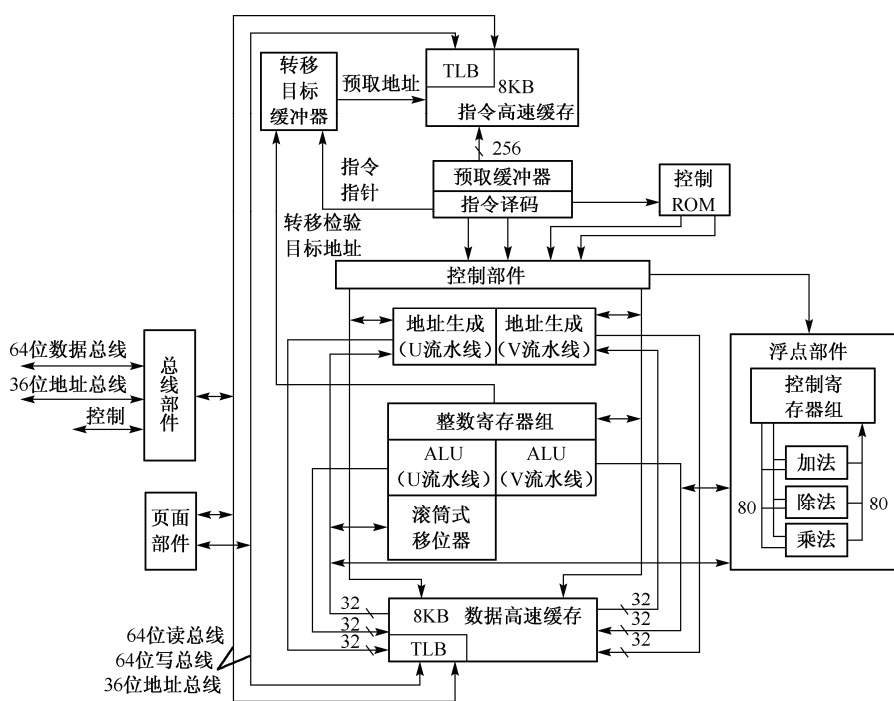


图 3-57 Pentium 微处理器的功能结构框图

用的浮点指令（加、减、除）采用专用硬件电路执行，而不像其他指令由微码来执行。因此，大多数浮点指令都可以在一个时钟周期内完成，这比相同频率下的 80486 浮点处理性能提高了 4 倍。

（3）指令转移预测部件

大多数情况下，程序指令的执行是一条指令接着一指令顺序进行的。指令流水线正是利用了这个特点，在同一时刻内，多个部件同时操作并形成流水线，这样可提高指令执行的吞吐量。但是程序中不时也有转移执行情况，即下一条指令需从另一存储区获取并执行，转移执行指令会冲掉流水线已有的内容，并重新装载指令流水线，这样会降低流水线效率和指令执行速度。如果微处理器知道何时发生转移和跳转的目标地址，就可不暂停流水线的操作，处理器的执行速度才不会降低。

Pentium 微处理器提供了一个小型的 1 KB 缓存(称为转移目标缓冲器 BTB)来预测指令转移。它用来记录正在执行的程序最近所发生的几次转移，这就好像一张指令运行路线图，指明转移指令很可能会引向何处。BTB 将进入流水线的新指令与它所存储的有关转移的信息进行比较，以确定是否将再次执行转移。如果找到一次匹配（BTB 的特征位命中），就产生一个目标地址，提前指出要发生的转移。如果预测正确，就立即执行程序转移，这样就不需要计算下一条指令的地址，而且可防止指令流水线停顿。反之，如果预测错误，将冲掉流水线中的内容，重新取入正确的指令，但这会有 4 个时钟周期的延迟。由于程序局部性原则，使得转移预测部件在大多数情况下能够正确预测，这就足以将微处理器的性能提高不少。

（4）数据和指令高速缓存

Pentium 芯片内部有两个超高速缓冲存储器，一个是 8 KB 的数据 Cache，另一个是 8 KB 的指令 Cache，它们可以并行操作。这种分离的高速缓存结构可减少指令预取和数据操作之间可能发生的冲突，提高微处理器的信息存取速率。

数据 Cache 除具有 80486 Cache 的通写方式外，还增加了数据回写方式，即 Cache 数据修改

后，不是立即写回主存，而是推迟到以后写入。这种延迟写入主存的方式减少了片内高速缓存与主存交换信息所占用的系统总线的时间，这对于多处理器共享一个公共的主存时特别有价值。当然，具有回写方式的数据 Cache 需要更复杂的 Cache 控制器。

(5) Pentium 的内部寄存器

Pentium 微处理器对 80486 的寄存器做了如下扩充。EFLAGS 标志寄存器增加了 2 位：VIF(位 19)、VIP(位 20)，它们用于控制 Pentium 虚拟 8086 方式扩充部分的虚拟中断。控制寄存器 CR0 的 CD 位和 NW 位被重新定义以控制 Pentium 的片内高速缓存，并新增了 CR4 控制寄存器对 80486 结构进行扩充。EFLAGS 还增加了几个模式专用寄存器，用于控制可测试性、执行跟踪、性能监测和机器检查错误等。

3.8.4 Alpha 微处理器

数字设备公司 (DEC) 在 20 世纪 60 年代后期推出的 PDP-11 系列，曾是当时最具代表性的 16 位小型机，至今仍在许多教科书中作为一种背景机型，用来阐明指令系统组成等基本原理。

1977 年，DEC 公司开始推出 32 位的超级小型机 VAX-11 系列，继续作为 32 位小型机的“首席”代表。随着微处理器技术（存取速率、访存空间、字长）的迅速发展，近年来传统的 VAX-11 技术已显落后。因此，DEC 公司于 1992 年推出了一种 64 位的高速 RISC 微处理器芯片 Alpha，一方面用它构造 64 位的工作站，另一方面用来改造传统的 VAX-11 系列高档机。由于 Alpha 的高性能和具有继续扩展的结构潜力，现正成为计算机工业界的一个热点，许多公司和厂家正竞相采用 Alpha 芯片制造并行处理系统，甚至最著名的巨型机制造者 CRAY 公司也尝试用多个 Alpha 芯片改造其巨型计算机。因此，我们选择它作为第四种典型加以简略介绍，预计它将成为 64 位机中的一个重要代表。

Alpha 芯片的内部结构框图如图 3-58 所示，其主要性能如下：

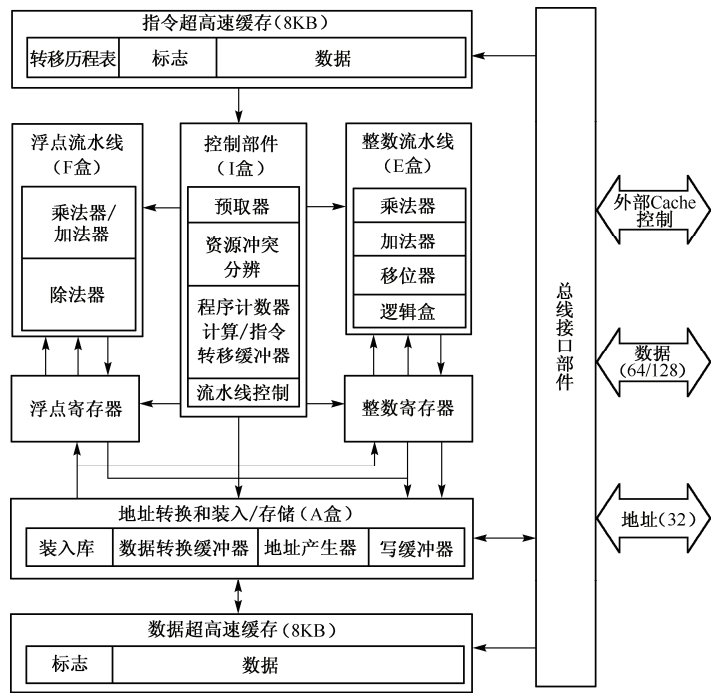


图 3-58 Alpha 芯片的内部结构框图

- ◎ 32 位物理地址，可直接寻址的物理存储空间为 4 GB。
- ◎ 64 位虚拟地址，使虚拟存储空间可达 16×10^{18} B。
- ◎ 分别有 8 KB 的指令超高速缓存与 8 KB 的数据超高速缓存。
- ◎ 字长 64 位，外部数据通道 64/128 位。
- ◎ 片内时钟频率 200 MHz，外部时钟频率 400 MHz，峰值速度 400 MIPS。

(1) 片内超高速缓存

片内分设两个超高速缓存。一个用于指令缓存，包含指令转移历程表、标志与指令代码（泛称为数据），命中率很高。另一个用于数据缓存。相对于 80486，Alpha 的片内高速缓存能力更强，还允许在片外配置高速缓存。

(2) 三个功能部件

① 整数部件——称为 E 盒，即常规定点运算部件，包含加法器、乘法器、移位器、逻辑运算部件（除法运算可转换为乘法进行）。

② 浮点部件——称为 F 盒，即浮点运算器，包含加法器、乘法器和专门的浮点除法器。

③ 地址转换和装入/存储部件——称为 A 盒，负责将整数/浮点数装入整数寄存器/浮点寄存器，或者将寄存器中的数写入数据超高速缓存。

(3) 控制部件

控制部件称为 I 盒，同时采用了流水线技术与超标量技术。Alpha 采用多级流水，并分设两条流水线：整数流水线与浮点流水线，从预取指令，进行资源冲突的分析，通过流水线控制，使指令按流水处理方式执行。超标量技术指可以同时执行几条数据无关联的指令，Alpha 在一个时钟周期内可以并行执行两条 32 位长指令，它可将两条指令分配到功能部件中去执行（整数存储和浮点操作，或者浮点存储和整数操作不能同时执行）。

(4) 总线接口部件

总线接口部件的基本功能与前述几种 CPU 相似，但 Alpha 的总线接口部件允许用户配置 64 位或 128 位的外部数据通道，调整所需要的外部超高速缓存容量和访问时间，控制总线接口部件的时钟频率，使用 TTL 电平或是 ECL 电平。

Alpha 的结构可扩展性很好，现正发展为一种系列。其低档芯片已进入个人计算机领域，而高档芯片发展潜力很大。DEC 公司乐观地认为，这一系列可望有 25 年的生存期，未来的 Alpha 芯片可以并行地执行更多条指令，其性能将提高 1000 倍，运算速度可望高达每秒执行 4000 亿条指令。

3.8.5 CRAY-1

CRAY 公司于 1976 年推出的 CRAY-1，是早期巨型计算机的公认代表。与传统小型机及后来出现的微型计算机的主要不同在于，它具有向量运算能力（或称为向量机），采取模 16 的多存储体交叉访问方式。CRAY-1 的体系结构一直延续使用在后来的 CRAY 系列巨型机中，其结构框图如图 3-59 所示。

(1) 主要组成

① 指令缓冲器，也就是前述的指令队列，图中共有 4 个指令缓冲器，每个为 64×16 位字段。

② 地址寄存器组 A 和地址服务寄存器 B。地址寄存器组 A 有 8 个 ($A_0 \sim A_7$) 24 位寄存器，用于提供主存地址、变址移位计数、循环控制、I/O 通道地址等。地址服务寄存器 B 是一种中间缓冲寄存器，由 64 个 ($B_0 \sim B_{63}$) 24 位寄存器组成，B 寄存器与主存间以数据块方式进行数据传

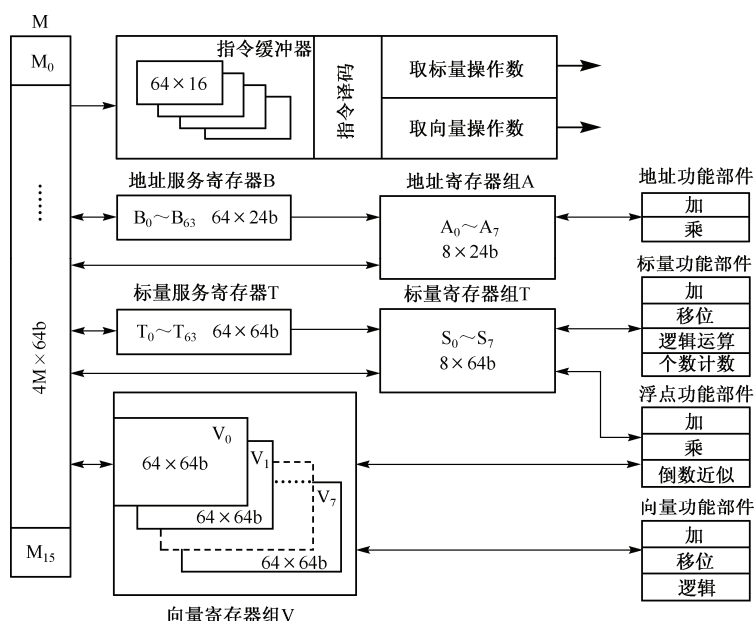


图 3-59 CRAY-1 内部结构粗框图

输。主存可以直接与 A 寄存器进行传送，也可先送入 B 寄存器，再由 B 向 A 传送。因此，地址服务器 B 的作用类似于高速缓存 Cache。

③ 标量寄存器组 S 和标量服务寄存器 T。标量寄存器组有 8 个 64 位的寄存器 $S_0 \sim S_7$ ，提供参与标量运算的操作数，保存操作结果。标量服务寄存器 T 的作用与 B 相似，提供缓冲，有 64 个 ($T_0 \sim T_{63}$) 64 位寄存器。主存可直接与 S 进行传输，也可通过 T 向 S 传输，主存与 T 之间也按数据块方式进行传输。

④ 向量寄存器组 V 有 8 个，即 $V_0 \sim V_7$ ，每个向量寄存器可存放一个 64 分量 \times 64 位的向量操作数。进行向量操作时，指令应规定向量长度 V_L 。由于每个向量寄存器只有 64 个分量，如果 V_L 大于 64，则需在程序控制下将它们分成若干组。向量寄存器以数据块方式直接与主存进行传输。

⑤ 12 个功能部件。运算功能部件共有 12 个，分为 4 组。

- ⊙ 标量功能部件有标量加法器、标量移位器、标量逻辑运算部件、个数计数器。个数计数器用来统计操作数在 1 之前有多少个 0。

- ⊙ 向量功能部件有向量加法器、向量移位器、向量逻辑运算部件。

- ⊙ 浮点功能部件有浮点加法器、浮点乘法器、倒数近似部件。两数的乘法先将操作数转换为浮点格式，由浮点乘法器完成；除法在转换为浮点格式后，利用倒数近似部件转换为乘法进行。

- ⊙ 地址功能部件有地址加法器、地址乘法器。

⑥ 主存储器。主存储器由 16 个体组成，采用多体交叉访问方式以解决存储瓶颈问题，容量 $4M \times 64b$ (32 MB)。每个存储体的存取周期为 50 ns，占 4 个时钟周期。

(2) 流水线

指令部件采用 3 级流水，即取指、指令译码、取数。

标量流水线最多可达 8 级，可分别对应于 8 个标量寄存器的分级流水式操作。此外，向量流水线最多也可达 8 级。

总的来看，流水线有 5 条：指令流水、地址流水、标量流水、向量流水和浮点流水。

3.8.6 Transputer

当前计算机系统结构发展的趋势之一是多处理器化，即利用超大规模集成电路技术制造出结构紧凑的处理单元。这种单元的处理功能不一定很强，但更侧重于支撑多个单元之间的连接与通信，可用多个单元构成多机并行处理系统。在这方面，英国 INMOS 公司的 Transputer 是一类风格独特的 RISC 芯片，我们选它作为另一种典型 CPU 代表。Transputer 一词由 Transistor 和 Computer 两词的合成而来。每个芯片是一个独立的算元，可作为一个基本结点。用若干个算元构成一个多处理器的并行处理系统，就像用若干个晶体管构成计算机系统一样。

Transputer 现已成为一种系列产品，硬件芯片有 32 位微处理器、浮点处理器、磁盘控制器以及图形控制器等。下面以 32 位的 Transputer 芯片 T414（如图 3-60 所示）为例，简要介绍其功能和内部结构。

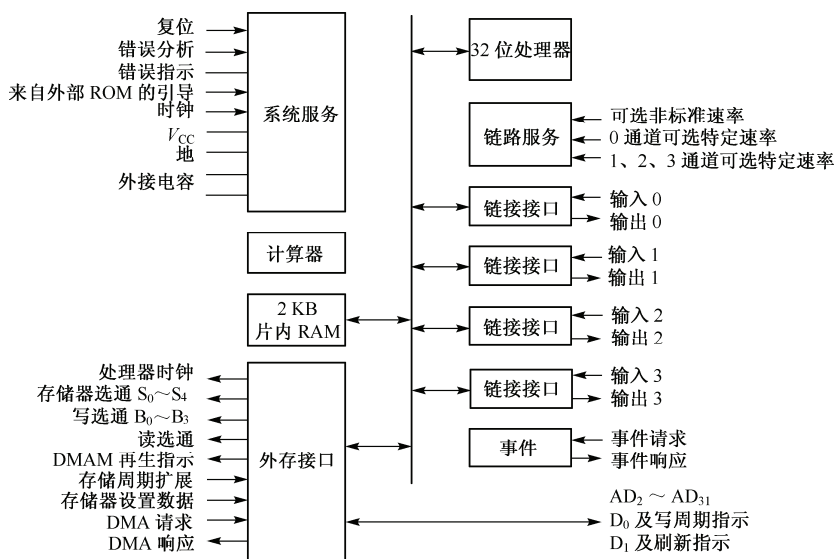


图 3-60 T414 功能模块示意图

① 32 位处理器。Transputer 具备一般 CPU 的基本功能，但指令格式简单，基本指令只有 1 字节，高 4 位为操作码，低 4 位为数据码。通过与操作数寄存器一道工作，可使实际操作数/地址大于 4 位。通过堆栈操作，也可扩展数据位数。通过前缀功能，可大大扩展指令数目。速度为 10 MIPS。

② 链接接口。T414 最重要的特点是具有 4 个 32 位的全双工通信链路（Link），其速率可以选择为 5、10、20 Mbps。所以，我们可以方便地实现多结点之间的链接，如以简单的结构实现一种 4 CPU 加速卡，并具有与其他加速卡连接的通信链路。

③ 片内 RAM。片内设有 2 KB 的高速 RAM，存取周期 50 ns（新产品速度更高）。处理器部分只设置了少量寄存器，工作时尽量利用这 2 KB 的高速 RAM。

④ 外部存储器的接口。T414 芯片的直接访存空间可达 4 GB，相应地，该芯片对外提供 32 位地址线/数据线。

⑤ 事件，可接收外部事件的请求，输出对外部事件的响应信号。

⑥ 系统控制，接收外部输入的有关控制信号，进行相应的处理，输出错误指示信号等。

为了开发 Transputer 系统，专门推出一种并行处理语言 OCCAM，作为这类系统的程序设计语言。它的结构设计和 Transputer 的机器语言类似，因而运行效率很高。硬件、软件配套也是这一体系的重要特色。

习 题 3

1. 简要解释下列名词术语

CPU 运算器 控制器 通用寄存器 暂存器 指令寄存器 IR 程序计数器 PC
 程序状态字 PSW 时序系统 微命令 组合逻辑控制 微程序控制 地址结构
 显地址 隐地址 寻址方式 立即寻址 直接寻址 寄存器寻址 间接寻址
 寄存器间址 间址单元 变址寻址 基址寻址 相对寻址 页面寻址 堆栈 栈顶
 堆栈指针 CISC RISC 全加器 并行加法器 进位链 串行进位 并行进位
 分组进位 指令周期 工作周期 时钟周期 微指令周期 总线周期 主存读/写周期
 微指令 微程序 控制存储器 增量方式 断定方式 SMT 超线程 多核

2. 简化地址结构的基本途径是什么？

3. 减少指令中一个地址信息的位数的方法是什么？

4. 某主存储器部分单元的地址码与存储器内容对应关系如下：

| 地址码 | 存储内容 |
|-------|-------|
| 1000H | A307H |
| 1001H | 0B3FH |
| 1002H | 1200H |
| 1003H | F03CH |
| 1004H | D024H |

(1) 若采用寄存器间址方式读取操作数，指定寄存器 R_0 的内容为 1002H，则操作数是多少？

(2) 若采用自增型寄存器间址方式 $(R_0)+$ 读取操作数， R_0 内容为 1000H，则操作数是多少？指令执行完后 R_0 的内容是多少？

(3) 若采用自减型寄存器间址方式 $-(R_1)$ 读取操作数， R_1 内容为 1003H，则操作数是多少？指令执行完后 R_1 的内容是多少？

(4) 若采用变址寻址方式 $X(R_2)$ 读取操作数，指令中给出形式地址 $d=3H$ ，变址寄存器 R_2 内容为 1000H，则操作数是多少？

5. 对 I/O 设备的编址方法有几种？请简要解释。

6. I/O 指令的设置方法有几种？请简要解释。

7. 用 74181 和 74182 芯片构成一个 64 位 ALU，采用分级分组并行进位链结构。画出逻辑图，并注明输入、输出信号。

8. 试比较组合逻辑控制和微程序控制的优缺点及应用场合。

9. 根据模型机数据通路结构，拟定 MOV 指令流程在 ST_2 、 ST_3 、 ST_4 中的操作时间表。

10. 拟出下述指令流程及操作时间表。

- | | | |
|------------------------|--------------------------|-------------------------|
| (1) MOV $(R_0), (SP)+$ | (2) MOV $(R_1)+, X(R_0)$ | (3) MOV $R_2, (PC)+$ |
| (4) MOV $-(SP), (R_3)$ | (5) ADD $R_1, X(R_0)$ | (6) SUB $(R_1)+, (R_2)$ |
| (7) AND $-(R_0), R_1$ | (8) OR $R_2, (R_0)+$ | (9) EOR $(R_0), (R_1)$ |
| (10) INC $X(PC)$ | (11) DEC (R_0) | (12) COM $(R_1)+$ |

- | | | |
|-------------------|------------------|------------------|
| (13) NEG $-(R_2)$ | (14) SL R_0 | (15) SR R_3 |
| (16) JMP SKP | (17) JMP R_0 | (18) JMP $X(PC)$ |
| (19) RST $(SP)+$ | (20) JSR (R_1) | |

11. 拟出中断周期 IT 中各拍的操作时间表。

12. 编写取目的地微子程序（从 60H 单元开始）。

13. 根据表 3-10 微程序，以微地址序列形式（如 00-01-02-0C...），拟出下述指令的读出与执行过程。

- | | | |
|-------------------------|--------------------------|-----------------------|
| (1) MOV $(R_0), (SP)+$ | (2) MOV $(R_1)+, X(R_0)$ | (3) ADD $X(R_0), R_1$ |
| (4) SUB $(R_1)+, (R_2)$ | (5) NEG $-(R_2)$ | (6) JMP (R_0) |
| (7) JSR R_1 | | |

14. 如果以 R_3 为堆栈指针，软件建立堆栈，试分别编写压栈及弹出操作的子程序。

15. 如果将 CPU 时钟周期与访存周期分开设置，一个访存周期占用 4 个时钟周期，试重新设计模型机指令流程。

16. 如果将模型机内部数据通路结构改为图 3-3 所示的结构，试重新设计模型机（给出总线与数据通路结构，拟定各类信息的传送途径，设置微命令）。

17. 简述指令流水线的工作原理。

18. 试比较超标量和超流水的异同点。

第 4 章 存储子系统

存储器是记忆信息的实体,是数字计算机具备存储数据和信息能力、能够自动连续执行程序、进行广泛的信息处理的重要基础。在传统的 CPU 中,为数不多的寄存器只能暂存少量信息,绝大部分的程序和数据需要存放在专门的存储器中。信息存储的机制有多种,采用的技术也非常广泛,不同的技术在存储器的组织方式、性能上都有很大不同,为计算机系统的不同需求提供服务。因此,计算机存储系统的组织是多层次、多结构的,这就构成了有机联系的存储子系统。

本章将介绍当前广泛应用的各类存储器的存储原理和存储系统的组织方式,重点讨论用半导体存储器构成的主存储器,包括 ROM、DRAM 和 SRAM 等,然后介绍现代存储器技术中提高存储器访问速度的高速缓存、虚拟存储器、并行存储器等有关技术。

4.1 概述

从不同的角度分析,计算机的存储系统表现出的特性也有所不同。例如,从物理构成角度,着重于整个存储系统是如何分级组成的;从用户调用角度,关心有哪几种存取方式;从存储原理(物理机制)的角度,讨论各类存储器的记忆信息原理。相应地,存储系统的各层次关心的是应当选用哪种存储器。如果说磁盘、磁带、光盘等外存储器需由专门的厂家生产,而在我们的实际工作中则常常需要自行设计半导体存储器(用存储芯片组成),那么,从设计者的角度应当如何设计?用哪些技术指标评价存储器的性能?

4.1.1 存储系统的层次结构

存储系统特别是主存储器与 CPU 之间有大量的信息交换操作,因此对存储器最基本的要求是:存储容量大、存取速度高、成本低价格低。系统存储器的容量越大,可存储的信息就越多,计算机的处理能力就越强。而且,计算机系统的大量处理功能都是通过执行指令完成的。因此,CPU 要经常从主存储器读取指令和数据,并回存处理结果。如果存储器不具备存取速度高的特性,计算机的整体性能就会受到很大的影响。随着计算机功能的迅速增加,需要执行的程序量日益增大,需要处理的数据量也越来越大。特别是应用于信息管理和知识处理的计算机,需要存储的信息量非常庞大。要想提高计算机的工作速度,存储器的存取速度是关键(常因存储器速度不满足要求而形成所谓瓶颈)。最后,针对具体应用的计算机系统,需要综合考虑系统构成的成本价格,以期获得最优的性价比。

在同样的技术条件下,上述这些要求往往是相互矛盾的,彼此形成制约,在同一个存储器中通常难以同时满足这些要求。在价格、容量、存取时间上往往存在以下“两难”关系:存取速度越快,每位的价格又越高;存储容量越大,每位的价格却越低;存储容量越大,存取速度又越慢。

要扩大半导体存储器的容量,一般要增加元器件的数量才能实现,但元器件的增多又会导致电路连线上的分布电容增大,从而使存储器的工作速率降低。磁盘的数据容量通常要比半导体存储器的容量大很多,但它依靠盘片的旋转来依次存取信息,其存取速率很难与半导体存储器相比。一般来说,生产高速存储器,其成本自然要高于低速存储器。

因此,技术的发展一方面需要寻求新的存储机理,努力改进制造工艺,以提高存储器的性能;另一方面,可以采用存储器分层结构来满足计算机系统对存储器不同方面的要求,而不仅依靠单一的存储部件或技术。让 CPU 直接访问的一级,速度尽可能快,而容量可以相对有限;作为后援的一级则容量应较大,其速度可以相对慢些。经过合理的搭配组织,用户就能够使整个存储系统

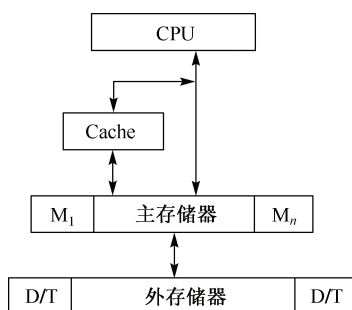


图 4-1 分层存储体系结构示意图

提供足够大的存储容量和较快的存取速度。

图 4-1 是典型的三级存储体系结构,可以划分为“高速缓冲存储器(Cache)一主存一外存”三个层次。现代计算机系统的存储结构基本形态仍是这三个层次,但高速缓冲存储器已进一步扩大,构成了如下存储体系架构:CPU 内部寄存器组→CPU 内部的第一级 Cache(L1)→第二级 Cache(L2)→甚至第三级 Cache(L3)→主存储器→外部辅助存储器(如硬盘和光盘等)。

在早期的低档微型计算机系统中,如 80286、80386 微机,可能只有主存和外存两个层次。在单板微机(一般不称为系统)中,或在多机系统的局部节点之中,可能只有一级半导体缓冲存储器与微处理器相配。目前在主流的计算机中,CPU 内部一般集成了至少两级 Cache(即 L1 和 L2)。在一些高端的多核 CPU 中,除了两级 Cache 外,还集成了第三级 Cache(L3),如 Intel 的 Itanium、Core i3/i5/i7 系列、AMD 的 Phenom II 系列。

在图 4-1 中,各级存储器按层次结构自上而下具有以下特征:① 存储器的每位(bit, 比特)价格逐渐降低;② 存储器的容量逐渐增大;③ 存储器的存取速度逐渐变慢。

1. 主存储器

主存储器是能由 CPU 直接编程访问的存储器,用来存放 CPU 当前执行所需要的程序和数据,通常与 CPU 位于同一主机范畴之内,常被称为“内存”。

对于大的任务,需要运行的程序和数据可能很多。在程序的编译、调试和运行过程中,也可能需要使用大量的软件资源。但是,在某一段时间内,CPU 执行所需的程序和数据只是其中一小部分,其余大部分暂时不会使用到。因此,可将当前即将运行的程序和数据调入主存,其他暂时不运行的程序与数据则暂存在磁盘等外部存储器中,并根据需要进行实时替换。从用户的角度看,这种调度替换是以文件为单位进行组织的。

为了满足 CPU 对其编程直接访问的需要,一般对主存储器有一些基本要求:

① 采用随机访问方式,随机访问的含义将在 4.1.3 节中说明。

② 工作速度要足够快。早期的动态存储器(DRAM)平均访问时间约为 100 ns,随后出现的同步动态存储器(SDRAM)的平均访问时间约为 1~10 ns,速率提高了 10 倍左右。目前主流的 DDR 系列存储器,如 DDR2 等,其平均访问时间又在 SDRAM 的速度基础上再次提升了数倍。在主存速度不断提高的同时,CPU 的速度也在不断提高,且相比主存速度的提高而言,CPU 的速度提高得更快。单从 CPU 的主时钟频率来看,从早期 8086 的 4.77 MHz 到 Intel Pentiumm 4 的 3.8 GHz,性能提高了 800 多倍。由此可见,主存速度还远低于 CPU 速度不断发展的需要。为此,更高端计算机系统中普遍采用了多存储体交叉访问的工作方式,使存储系统的整体速度在宏观上能与 CPU 的处理速度基本匹配。

③ 具有一定的存储容量。计算机系统主存的容量和存取速率对程序的运行都具有重要的影响。如果主存容量过小,CPU 将很难有效地运行大规模程序,因为频繁地在主存与外存之间交换

数据,会增加系统开销,使效率严重下降。从物理寻址角度来看,主存的容量受地址位数的制约。如 8 位机能提供 16 位地址,可直接编程访问的主存空间只有 64 K。16 位机经过扩展,一般能提供 20 位地址,直接寻址空间可达 1 M。32 位机(80386/80486)可提供 32 位地址,理论上其直接寻址空间可高达 4 G。一个计算机系统究竟需要配置多大容量的主存取决于系统的设计规模,即根据需求和成本这两方面来进行综合考虑。此外,如前所述,容量和存取速率指标往往有一定矛盾,很难兼得。

主存储器一般由动态随机存储器(DRAM)组成,其特点是单片存储容量大,但存取速率较慢,且需要动态刷新。常用的动态随机存储器主要有:早期的快速页面模式 DRAM(FPM DRAM),具有检错纠错功能的 DRAM(ECC DRAM),扩展输出的 DRAM(EDO DRAM),同步 DRAM(SDRAM),以及后续发展起来的双倍数据率 SDRAM(DDR SDRAM)等。现代微型机系统的主存都已普遍采用 DDR2 和 DDR3 SDRAM 技术,以上技术将在 4.3.4 节中介绍。

除了大量采用动态随机存储器的主存外,计算机系统中还有少量用于保存固化程序和数据只读存储器(ROM),这些只读存储器通常用 EPROM、E2PROM 或 FLASH 来充当,现代微型机系统中一般使用 ROM 来存储固化程序和数据,如 BIOS 中的存储芯片,这些只读存储器中保存的程序和数据一般只在系统启动时才会调用运行。

2. 外存储器(辅助存储器、后援存储器)

由于主存容量有限(受地址位数、成本、存取速率等因素制约),大多数计算机系统中还设置了另一级大容量存储器,如磁盘、磁带、光盘等,作为对主存的后援和补充。它们位于传统主机的逻辑范畴之外,常被称为外部存储器,简称外存。

程序和数据只有进入到主存中才能被 CPU 运行,而位于外部存储器中的程序和数据则不能被直接运行。外部存储器主要用来存放需要联机保存但暂不使用的程序和数据。计算机系统虽然提供了丰富的软件,如操作系统、编译程序、文字编辑程序和调试工具软件等。但某个用户可能只需使用其中的一部分,或者在工作的某个阶段暂时只使用其中的一部分,如在编程时只使用到代码编辑软件、编译时只使用到编译程序、调试时只使用到调试软件等,所以可将各种软件资源存放在磁盘中,需要用到某个文件时才将它调入主存中,使用完毕再将其写回磁盘保存,以便将主存空间留给当前急需的程序和数据。有些程序和数据可能比较庞大甚至超出了主存的存储容量极限,我们也常将它存放在磁盘中,只将当前需要运行的程序和数据等调入主存。

根据外部存储器担负的主要任务,它应当具有很大的存储容量。相对于主存,对外存储器的速率要求则要低一些,通常存储一位的平均成本(即价格/位)也比主存要低很多。许多外部存储器的存储介质,如移动硬盘、U 盘和光盘等,在记录信息后可以脱机保存,需要联机使用时才将外部存储器接入计算机系统。

外存储器本身可以是多台独立的存储器,也可以分级构成。例如,将调用频繁的信息保存于磁盘存储器中,作为主存的直接后援;将调用不太频繁的信息保存于 U 盘或者移动硬盘存储器中,作为磁盘的后援,构成“主-辅”外部存储体系。为了提高访问磁盘、光盘的响应速度,常用高速半导体存储器构成缓冲存储器,其介于外存与主存之间。有些系统采用了磁盘缓冲存储结构后,其磁盘调用的平均响应时间能减少约 40%。

如前所述,CPU 不能直接使用存放在外存中的程序和数据,需要先将磁盘、光带等外存中的程序与数据调入主存之后,CPU 才能运行它们。对于主存,CPU 可按字或字节访问、处理。对于外存,用户编程往往是按文件名调用,以文件为一次调用的单位。但在物理存储结构中,常将一个文件分为若干数据块,CPU 在调用时可以数据块为单位调用。

3. 高速缓冲存储器

随着超大规模集成电路技术的发展,半导体存储器的存取速度已有很大提高。相比之下,CPU工作速率提高更快,两者之间一直存在约一个数量级的差距。实际上,每当有可能在单片芯片上集成更多的电路时,CPU的设计者总是用这些新技术来实现流水线和超标量运算,以便大幅提高CPU的速率。存储器的设计人员则利用这些技术来提高存储芯片的容量,而不是速率,这就使两者的速度差别越来越大。一个实际系统的主存必须满足一定存储容量的要求,而且受到制造成本的制约,这使得主存的速率不能很好地与CPU匹配。这种速率的差异使得在CPU发出访问存储器请求后,要经过多个CPU周期才能读到存储器的内容。

为了解决CPU与主存之间的速度匹配,许多计算机系统中设置了一种高速缓冲存储器(Cache),见图4-1,其速率几乎可以与CPU一样快。高速缓存中存放的是CPU最近要使用的程序和数据,作为主存中当前活跃信息的副本。

当CPU要访问一个数据时,一般首先在Cache中查找。通过对给出的主存地址码的分析可以判断出所访问的主存存储区的内容是否已复制到Cache之中。若所需访问的主存区间已经复制到Cache中,则直接从Cache中快速读取,称为Cache访问命中。若当前访问区间内容不在Cache中,称为Cache访问未命中,此时需从主存中读取信息,并将当前数据所在的数据块整体调入并更新Cache。为此,需要实现主存地址与Cache物理地址间的映射变换,并采取某种调度算法(策略)进行Cache内容的更新和替换。

Cache作为现代计算机存储器子系统的一部分,主要是为了缓解CPU与主存在速率上的不匹配,通常由存取速度较高的静态随机访问存储器(SRAM)来构成,受体积的限制使其容量不能做得很大,其存取周期一般可达10 ns内。缓存基本上都是采用静态随机访问存储器(SRAM)来构成。最早先的缓存容量很低,英特尔公司从Pentium时代开始把缓存进行了分类。当时集成在CPU内核中的缓存已难以满足CPU的需要,受工艺限制又不能大幅提高缓存容量,因此早期还出现过集成在主板上的缓存,但这种方式早已被淘汰。那时,集成在CPU内的缓存称为一级缓存(L1 Cache),外部的则称为二级缓存(L2 Cache)。一级缓存中还为分数据缓存(Data Cache, D-Cache)和指令缓存(Instruction Cache, I-Cache),两者分别用来存放数据和执行这些数据的指令,而且两者可以同时被CPU访问,减少了争用Cache所造成的冲突,提高了处理器性能。Intel在推出Pentium 4系列处理器时,还用了一种新增的一级追踪缓存替代指令缓存,容量为12 KμOps,其表示能存储12K条微指令。

随着半导体工艺的发展,二级缓存已逐渐被集成在CPU芯片内了,其容量也被大幅提升。再用集成在CPU内部与否来定义一、二级缓存已不恰当。而且随着二级甚至三级缓存被集成入CPU内中,以往二级缓存与CPU速度差距很大的情况也已被改变,此时二级缓存几乎能以接近于CPU主频的速率工作,可以为CPU提供更高效的数据传输。二级缓存是CPU性能表现的关键之一,在CPU核心不变化的情况下,增加二级缓存容量能使性能大幅度提高。同一内核数量的高低端CPU,其往往在二级缓存上也有差异,由此可见二级缓存对于CPU的重要性。目前,高速缓存的访问命中率可高达95%以上,这就使CPU从整体上能以接近高速缓存的速度访问存储器,而总存储容量又相当于联机外存的总容量。

对于高性能的CPU来说,高速缓存设计的重要性与日俱增,对于Cache的设计除了要提高其命中率外,还需从如下几方面加以考虑。第一,Cache的容量越大,越能满足CPU的性能要求,但成本也越高。第二,高速缓存数据块的大小,如64 KB的Cache既可分为1 K个数据块(64B/块),也可分为2 K个数据块(32B/块),同时还有其他分法。第三,Cache如何组织,即如何与

主存储器的字相对应。第四，指令和数据是共享一个高速缓存还是分别用不同的 Cache 存放。随着 CPU 中流水线技术的广泛使用，当前的趋势是采用分体缓存。第五，高速缓存的个数。目前，主流 CPU 内部集成了第一级 Cache 和第二级 Cache，一些高端处理器甚至集成了第三级 Cache，这部分内容在 4.6 节中将详细阐述。

4.1.2 物理存储器与虚拟存储器

虚拟存储器是与计算机存储子系统组织技术相关的一个重要内容。4.1.1 节已从物理存储结构上将计算机的存储子系统分为三个层次。此外，如果从用户的角度看，还可形成另一种存储子系统的层次结构观点，即物理存储器和虚拟存储器。

虚拟存储器是依靠操作系统和硬件的支持来实现的，能从逻辑上为用户提供一个比物理存储容量大得多，可寻址的“主存储器”。虚拟存储区的容量与物理主存大小无关，而受限于计算机的地址结构和可用磁盘容量。虚拟存储器只是一个容量非常大的存储器的逻辑模型，不是任何实际的物理存储器，它借助于磁盘等辅助存储器来扩大主存容量，可以为更大或更多的程序所使用。虚拟存储器面向的是主存-外存层次，它以透明的方式给用户提供了一个比实际主存空间大得多的程序地址空间。

虚拟存储器的主要思想是把可访问的逻辑地址空间和实际的物理内存空间区分开，通过某种技术手段将外存数据与内存区域关联，让用户通过较小容量的内存来访问到较大容量的外存数据。例如，若计算机的地址码为 32 位，则可直接寻址的地址空间为 4 G，还配备了数百 GB 的外部存储器，采用虚拟存储器技术，在操作系统和相关硬件的支持下，通过相应的技术策略，就可使用户能访问的外部存储器编址空间远大于 4 GB 的主存空间，这就会使用户感到自己在访问一个很大的存储器，但实际上的主存却还是只有 4 GB。

我们把计算机系统提供给用户透明访问的这个存储器，即在软件编程上可使用的存储器，称为虚拟存储器，其存储容量叫虚拟存储空间，简称虚拟空间。面向虚拟存储器的编程地址称为虚地址，也称为逻辑地址。真正在物理上存在的主存储器被称为物理存储器，简称为实存（与虚存相对应），访问主存的地址称为物理地址或实地址。

除了可寻址访问的存储空间远大于实际主存容量外，在物理实现上还为外部存储器（如磁盘）提供了硬件支持，可将当前暂时不用的信息存放在外存中。在软件方面，依靠操作系统实现主存与外存间的信息更换，只让当前需要使用的信息调入主存。这一更换过程对用户是透明的，因此用户感到可使用的编程空间很大。为了实现虚拟存储器，需将虚拟存储空间和物理实存空间按一定格式分区组织和管理，如页式管理、段式管理、段页式管理等，并提供虚/实地址的自动转换，将用户编程中提供的虚地址（逻辑地址）自动、快速地转换为实地址（物理地址），再根据该地址访问真实的主存储器。此外，需采取某种调度策略，实现主存内容的更换，使当前需要的程序和数据能够及时进入主存之中。

虚拟存储器是从用户界面上透明可见和可用的存储空间，并不是真实物理存在的，更不是外部存储器与主存的简单拼合。从编程的角度看，用户使用虚拟存储空间就如同使用主存一样，而内部的调度管理和信息交换则是由操作系统和硬件一起配合实现的。

4.1.3 存储器的分类

不同类型的存储器具有各自不同的特性。下面从存储机制、存取方式、读写特性、在系统中所起作用等方面讨论各类存储器的特点。

1. 物理存储机制（存储介质）

从物理机制上看，有许多种可供利用的存储原理。凡是明显具有并能保持两种稳定状态的物质和器件，如果能够方便地与电信号进行转换，就可以作为存储介质。这两种稳定状态或者它们之间的变化，可作为记录二进制代码 0 和 1 的基础。由电信号产生与二进制代码相应的记录状态，称为写入，即将信息写入存储介质；根据存储介质的状态产生相应的电信号，称为读出。

（1）半导体存储器

现在的主存储器普遍采用半导体存储器。利用大规模、超大规模集成电路工艺制成各种存储芯片，每个存储芯片包含多个晶体管，具有一定容量；再用若干块存储芯片组织成主存储器。半导体存储器又分为静态存储器和动态存储器两种。

① 静态存储器

静态存储器依靠双稳态触发器的两个稳定状态保存信息。每个双稳态电路可存储一位二进制代码 0 或 1，一块存储芯片上包含若干个这样的双稳态电路。双稳态电路是有源器件，需要电源持续供电才能保持电路状态稳定。只要电源正常，就能长期稳定地保存信息，所以称为静态存储器。如果电源断开，则电路的双稳态被破坏，其存储的信息将会丢失，属于挥发性存储器，或称为易失性。正是这个原因，尽管半导体存储器一开始就表现出优于早期的磁芯存储器的种种性能，如集成度高、容量大、速率快、体积小、功耗低等，但其易失性缺陷推迟了它的广泛应用，直到 20 世纪 70 年代才取代磁芯存储器。目前的措施是：如果需要在断电后保存信息，可采用低功耗半导体存储器，用可充电电池作为后备电源，当交流电源不正常时，立即自动切换到后备电源。

从集成电路类型划分，半导体存储器可分为双极型和 MOS 型两大类。双极型存储器的读写速率快（特别是 ECL 型）、功耗大、集成度低，适于作为小容量的快速存储器，又分为 ECL 型和 TTL 型，如高速缓存 Cache，或作为专门的集成化寄存器组。MOS 型可以分为 NMOS 和 CMOS。NMOS 存储器的工艺较简单，集成度高，功耗小，单片容量大，适于作为主存储器；CMOS（互补 MOS）存储器的功耗最小，在纽扣电池一类的后备电源供电下，可将存储信息保持数月之久，适于作为“不挥发性存储器”。

② 动态存储器

动态存储器是依靠电路电容上存储的电荷来暂存信息的。存储单元的基本工作方式是：通过 MOS 管（也称为控制管）向电容充电或放电，充有电荷的状态对应信息 1，放电后的状态对应信息 0。虽然电容上电荷的泄漏很小，但工艺上仍无法完全避免泄漏，时间一长，电荷会全部泄露，因此需要定时刷新暂存的内容，即对存 1 的电容补充电荷。由于需要动态刷新，所以也称为动态存储器。为了使电荷泄漏尽可能小，动态存储器多采用 MOS 工艺，因为 MOS 管和 MOS 电容的绝缘电阻极大，电容上电荷的保存时间较长。

动态存储器的内部结构简单，功耗也比较低，在各类半导体存储器中，它的集成度最高，适合用于大容量的主存储器。

（2）磁表面存储器

磁表面存储器是利用磁层上不同方向的磁化区域存储信息。磁表面存储器采用矩磁材料的薄膜，构成连续的磁记录载体，在磁头作用下，使记录介质的各局部区域产生相应的磁化状态，或形成相应的磁化状态变化规律，用以记录信息 0 或 1。由于磁记录介质是连续的磁层，在磁头作用下才划分为若干磁化区，所以称为磁表面存储器。

其存储体的结构，是在金属或塑料基体上，涂敷（或电镀、溅射）一层很薄的磁性材料，这层磁膜就是记录介质，或称为记录载体。根据其形状，磁表面存储器可分为磁卡、磁鼓、磁带、

磁盘。目前，磁带和磁盘是主要的外存储器。

磁表面存储器的存储容量大，且每位价格很低，非破坏性读出，信息保存期长。但其结构和工作原理决定了读写方式很特殊，即需要让记录介质做高速旋转或平移，磁头才能对其读写。这是机械运动方式，所以存取速率远低于半导体存储器，一般作为外存使用。

(3) 光盘存储器

光盘是利用光来存储的装置，它的出现是信息存储技术的重大突破。其基本原理是用激光束对记录膜进行扫描，让介质材料发生相应的光效应或热效应，如使被照射部分的光反射率发生变化，或出现烧孔（融坑），或使结晶状态变化或磁化方向反转等，用于表示 0 或 1。

① 只读型光盘（Compact Disk, Read Only Memory, CD-ROM），以烧孔（融坑）形式记录信息，由母盘复制而成，不能改写，提供固化的信息，如程序数据、图像信息、声音信息等，广泛用于多媒体技术中。

② 写入式（写一次型，Write Once Read Many, WORM）光盘可由用户写入信息，写入后可以多次读出，但只能写一次，信息写入后不能修改。这种光盘主要用于计算机系统上的文件存档或写入的信息不需修改的场合。

③ 可擦除/重写型（可逆式）光盘。激光束使介质产生的物理变化是可逆的，因而可以擦除重写。主要有两种记录原理：光磁记录（利用热磁反应）和相变记录（利用晶态-非晶态转变）。与其他类型的光盘相比，这种光盘的性价比不高，因此并未广泛流行。

2. 存取方式

从用户编程角度看，我们关心信息的存取方式，它影响到存储信息的组织。

(1) 随机访问存储器（Random Access Memory, RAM）

主存和高速缓存 Cache 是 CPU 可以直接编址访问的存储器，这就要求它们采取随机存取方式。随机存取的含义有两点：

- ⊙ 可按地址随机地访问任一存储单元，如可直接访问 0000H 单元，也可直接访问 FFFFH 单元；CPU 可按字节或字存取数据，进行处理。
- ⊙ 访问各存储单元所需的读写时间完全相同，与被访问单元的地址无关，一般可用读写周期（存取周期）来表明 RAM 的工作速度。

如前所述，按照所用存储器芯片类型，分为静态随机存储器和动态随机存储器两种。

(2) 顺序访问存储器（Sequential Access Memory, SAM）

顺序访问存储器的信息是按记录块组织且按顺序在介质上存放的，访问所需的时间与信息存放位置密切相关。磁带是一种采取顺序存取方式的典型存储器，当要访问其中的某个文件的某个数据块时，必须让磁带正向或反向走带，按顺序找到所需的文件数据块，并顺序地读出。写入的过程与此相似，需要顺序写入。所以，访问某个文件的时间视磁头与文件起始处的距离而定。这种顺序存取方式不适于主存，只能用于外部辅助存储器。虽然顺序存取速率较慢，但磁带的存储容量大，每位价格低。

(3) 直接访问存储器（Direct Access Memory, DAM）

直接访问存储器在访问信息时，先将读写部件直接指向某一小区域，再在该区域中顺序查找，访问时间跟数据所在的位置也密切相关。磁盘是一种典型的直接访问方式的存储器。在磁盘中，每个记录面划分为若干同心圆磁环（即磁道），每个闭合磁道中又分为若干个扇区，信息按位串行地记录于磁道中。按照这种信息分布结构，磁盘的寻址过程分为两个阶段：首先，磁头（读写部件）沿盘面径向移动，直接定位于某个磁道上；然后，磁头再沿磁道顺序地读写。因此，磁盘

的存取方式介于纯随机存取方式与纯顺序存取方式之间，由于它首先直接指向存储器中某个较小的局部区域，不必都从头开始顺序寻址，因而称为直接存取存储器，以区别于磁带那样的完全顺序存取方式。这种方式的存取时间也与信息所在位置有关，但快于顺序存取方式，所以适用于调用较频繁的外存，作为主存的直接后援，如硬盘。

3. 读写特性

从读写特性角度出发，存储器可以划分为可读可写型存储器（Read-Write Memory, RWM）和只读型存储器（Read-Only Memory, ROM）。

常见的可读可写型存储器有 RAM、磁盘等。

只读存储器在正常工作中只能读出，不能写入。主存中常采用一部分 ROM 来固化系统软件中的核心部分、已调试完毕不再更改的应用软件，以及汉字字库一类信息等。CPU 中也常采用 ROM，用来存放解释执行机器指令的微程序。这样的 ROM 虽采用随机访问存取方式，但由于其只读不写的特性，常被归划为专门的一类。

早期曾用磁环、二极管矩阵等构成 ROM，现在普遍采用大规模半导体集成电路。半导体集成电路型 ROM 又分为固定掩模型（用户不能写入）ROM、一次编程写入型 PROM、紫外线擦除可编程型 EPROM、电擦除可编程型 EEPROM、FLASH（闪存）等。

4. 存储器在系统中的位置

按存储器在计算机系统中所处的位置（或所起的作用），存储器又可分为主存储器、外存储器和高速缓冲存储器，它们的特点和作用已在 4.1.1 节中说明。

4.1.4 存储器的技术指标

对于存储器，我们关心的首要特性是存储容量，主存储器通常用字节或字表示，字长有 16、32 或 64 位等。外存储器的容量通常用字节表示。与容量相关的一个概念是传输单位，对于主存，传输单位是指每次读出或写入存储器的位数，通常等于字长；对于外存储器，传输单位一般为数据块。存储器的其他重要特性是与存取速率相关的 3 个参数。

（1）存取时间

信息存入存储器的操作称为写操作。从存储器取出信息的操作称为读操作。读、写操作统称为“访问”。从存储器接收到读（或写）申请命令到从存储器读出（或写入）信息所需的时间称为存储器访问时间（Memory Access Time）或称为存取时间，用 T_A 表示。存取时间 T_A 是反映存储器速率的指标，取决于存储介质的物理特性和访问机制的类型， T_A 决定了 CPU 进行一次读写操作必须等待的时间。

对于随机存储器，存取时间 T_A 一般是指从地址传送给存储器时起，到数据已经被存储或能够使用时止，这期间所需要花费的时间。目前，大多数计算机系统的随机存储器的存取时间一般都在纳秒（ns）级。

（2）存取周期

存储器的另一个重要的技术指标是“存取周期”（Memory Cycle Time），一般用 T_M 表示。 T_M 用来指示存储器连续进行两次完整的存取操作所需的最小时间间隔。存取周期主要是针对随机存储器的，具体是指本次存取的开始时间到下一次存取的开始时间之间的距离， T_M 即 T_A + 复原时间。因为有些存储器的读出操作是破坏性的，当读取信息后原存信息被破坏，所以在读出信息的同时要立刻将其重新写回到原来的存储单元中，然后才能进行下次读写操作。即使是非破坏

性读出的存储器，读出后也不能立即进行下一次读写操作，因为存储介质与相关的控制线路都需要有一段稳定恢复的时间。综合分析，存取周期应该被理解为存储器进行连续两次访问所能允许的最小时间间隔。

T_M 是反映存储器性能的一个重要参数，常被标记在内存芯片上，如“-7”、“-15”、“-45”分别表示 7 ns、15 ns、45 ns，其数值越小表明内存芯片的存取速率越高，一般其价格越贵。 T_A 和 T_M 的具体参数值可按内存型号查阅相关的技术说明书。存储器的存取周期一般比存取时间长，通常 $T_M > T_A$ 。

主存访问时间一般是固定的，不受被访单元在主存中所处位置的影响。但像磁带、磁盘一类的存储器其存取速率既取决于读写操作的时间，也取决于磁盘、光盘转动和寻找信息存储位置的机械运动时间。因为每次寻址的时间不相同，所以取它们的平均值表明访问速率的快慢，称为“平均存取时间”。磁盘的平均存取时间一般在毫秒（ms）级。

（3）数据传输率

数据传输率（Data Transfer Rate, DTR）是指单位时间内可向存储器中写入或从存储器中读出数据的数量，一般也可称为存储器的传输带宽。存储器的平均数据传输率（或带宽）通常定义为下列计算公式：

$$R = \frac{W}{T_M} \quad (4-1)$$

其中， R 表示存储器的数据传输率，单位通常为 KB/s 或 MB/s；存取周期的倒数 $1/T_M$ 表示单位时间（每秒，s）内能够读写存储器的次数（即频率）； W 表示存储器一次读写数据的位数（bit），也就是存储器传输数据的宽度（比特数）。

【例 4-1】假定一台计算机的显示存储器用 DRAM 芯片实现，若显示器的分辨率为 1024×768 ，像素的颜色深度为 24 位，屏幕的刷新频率为 80 Hz，计算需要的显存带宽。

显示 1 帧需要从显存中传输的数据量： $W = 1024 \times 768 \times 24$ （b）。

显示 1 帧所需的数据传输时间： $T = 1/80$ （s）。

已计算得到显存每次传输数据大小和传输时间，根据前述式（4-1），则

$$R = \frac{W}{T} = \frac{1024 \times 768 \times 24}{\frac{1}{80}} = 1024 \times 768 \times 24 \times 80 \text{ b/s} = 180 \text{ MB/s}$$

4.2 半导体存储原理

由于半导体存储器具有很高的存取速率、较大的存储容量等显著特点，因此它是计算机执行程序时存储程序代码和操作数的主要场所。目前，几乎所有主存储器都采用半导体芯片组成。下面简要介绍几种常用的半导体存储单元电路及其芯片的工作原理。用户关心的是芯片外特性及用芯片构成存储器，但对芯片内部的工作原理基础知识有一定的了解也是有益的。

4.2.1 双极型存储单元

双极型存储器有 TTL（Transistor-Transistor Logic）型和 ECL（Emitter Coupled Logic）型两种，工作速率快，但功耗大、集成度较低，适于作为小容量快速存储器，如高速缓存或集成化通用寄存器组。TTL 型存储单元可用两个双射极晶体管交叉反馈，构成双稳态电路；也可用两个单射极

晶体管构成双稳态电路。电路中,用肖特基抗饱和二极管控制双稳态电路与位线的通断,这种形式的存储器速率较快,目前约为 25 ns。ECL 存储器速率更快,可达 10 ns。

1. TTL 型存储单元举例

二极管集电极耦合式的 TTL 型存储单元电路如图 4-2 所示。这种存储单元的基本结构如下。

① 晶体管 VT_1 和 VT_2 , 通过彼此交叉反馈构成一个双稳态电路。② 发射极接字线 Z , 如果字线为低电平, 可进行读写; 如果字线为高电平, 则存储单元处于保持状态, 保持原存信息不变。③ 双稳态电路通过一对肖特基抗饱和二极管 VD_1 和 VD_2 , 与一对位线 \bar{W} 和 W 相连接; 读写时, VD_1 和 VD_2 导通, 位线与双稳态电路连通, 可以通过位线状态改变双稳态电路状态 (写入), 或从位线上检测出读出信号 (读出); 保持状态时, 位线与双稳态电路脱离, 双稳态电路依靠自身的交叉反馈维持原有状态。

定义 当 VT_1 通导而 VT_2 截止时, 存储信息为 0; 当 VT_1 截止而 VT_2 通导时, 存储信息为 1。这种存储单元有两种读写方式: 一种是单边读写方式, 让一根位线电平不变, 通过另一根位线电平变化, 以写入信息; 另一种是双边读写方式, 根据写 0 或者写 1, 分别改变位线 \bar{W} 或 W 电平, 以改变双稳态电路状态。由于读写都是通过位线进行的, 所以位线又被称为写驱动/读出线。

图 4-3 给出了双边读写方式的有关电平设置。

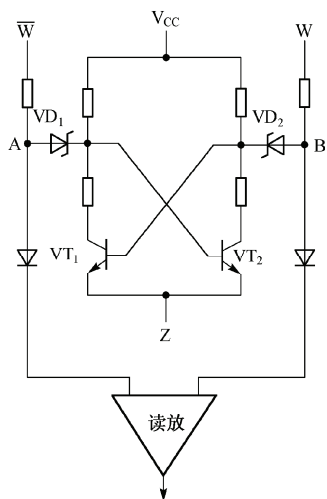


图 4-2 二极管集电极耦合式的双极型单元

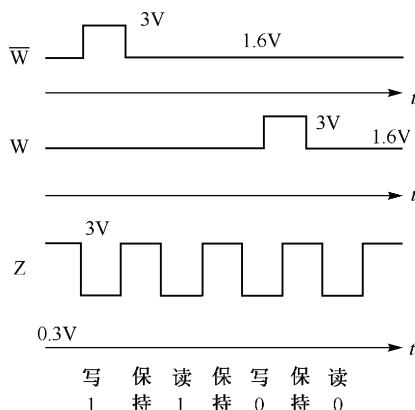


图 4-3 双极型存储单元的读/写方式

(1) 写入“1”

写入时, 字线 Z 加负脉冲, 其电平从 3 V 下降至 0.3 V。若要写入 1, 则位线 \bar{W} 的电平上至高电平 3 V, 而 W 线维持 1.6 V 不变, 于是二极管 VD_1 处于正向偏置而导通, 写入电流从 \bar{W} 线经 VD_1 流入 VT_2 的基极, 使 VT_2 导通。 W 线与 Z 之间的电平差远小于 \bar{W} 与 Z 之间的电平差值, 且 VT_2 通导, 经交叉反馈后将使 VT_1 截止。

(2) 信息保持

在存储电路保持状态中, 字线为高电平 3 V, 而位线 \bar{W} 、 W 均为 1.6 V, 则 VD_1 和 VD_2 均处于反偏状态而截止, \bar{W} 和 W 这一对位线与双稳态电路隔离不通, 因此 VT_1 、 VT_2 通过交叉反馈后维持原态不变。

(3) 读出“1”

读出时, 位线 \bar{W} 、 W 保持 1.6 V, 字线加电平为 0.3 V 的负脉冲。如果原存信息为 1, 即

VT₁截止、VT₂通导，则 W 线经 VD₂到 VT₂有较大电流通过；而 VT₁不通，则 \overline{W} 线上基本无电流。通过读出放大器将 W 线上的信号放大，可检测出原存信息为 1，称为“读 1”。

(4) 写入“0”

若要写入 0，则字线加电平为 0.3 V 的负脉冲，W 线电平上升为 3 V， \overline{W} 线维持 1.6 V 不变。于是二极管 VD₂导通，写入电流从 W 线经 VD₂流入 VT₁基极，使 VT₁导通，通过交叉反馈又使 VT₂截止。

(5) 读出“0”

读出时，Z 线加 0.3 V 的负脉冲， \overline{W} 、W 线维持 1.6 V 不变。若原存信息为 0，则 VT₁是通导的，故 \overline{W} 线经 VD₁到 VT₁有较大电流通过，又因 VT₂是截止的，故 W 线基本无电流。读出放大器将 \overline{W} 线上信号放大，可检测出原存信息为 0，称为“读 0”。

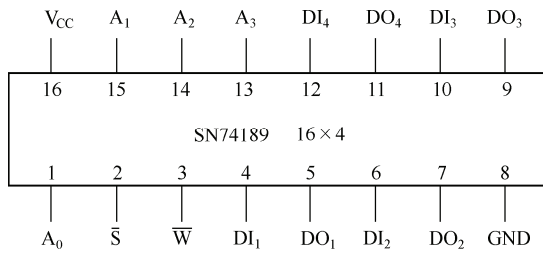


图 4-4 SN74189 芯片引脚图

2. TTL 型存储器芯片举例

为了便于理解，以通用的 74 系列中一种小容量存储芯片 SN74189 为例，介绍 TTL 型存储芯片的基本情况，如图 4-4 所示。

(1) 引脚及功能

SN74189 为 16 脚双列直插式封装，存储容量为 16 单元×4 位。注意，在讨论存储信息的物理

机制时，存储单元通常是指存储一位二进制代码的单元；但在讨论芯片外特性时，存储单元通常是指从使用角度的一个编址单位，如本例中一个单元含 4 位。

电源 V_{CC} ，接+5 V；地，GND；片选 \overline{S} ，为低电平时选中芯片，使能工作；地址 4 位 $A_3 \sim A_0$ ，可选择片内 16 个单元之一；数据输入 $DI_4 \sim DI_1$ ，数据输出 $DO_4 \sim DO_1$ ；读写命令 \overline{W} ，引脚低电平时写入，高电平时读出。

(2) 内部结构

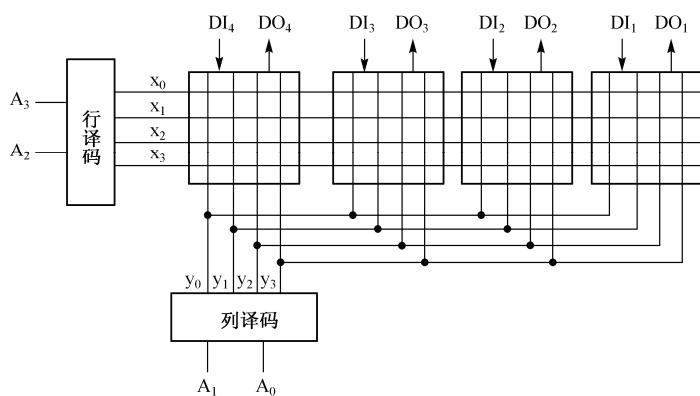
如何将图 4-2 所示的存储单元组织为一个存储芯片，使其呈现图 4-4 那样的外特性呢？显然，SN74189 中含有 $16 \times 4 = 64$ 个物理存储单元。由于每个编址单元有 4 位，所以将 64 个单元组成 4 个位平面。每个位平面包含一个 $4 \times 4 = 16$ 的矩阵，对应于 16 个编址单元，如图 4-5(a)所示。

图 4-5(b)是存储芯片内一个位平面的逻辑结构示意图。在位平面中，16 个单元排列成 4×4 矩阵，即分为 4 行、4 列。高位地址 A_3 、 A_2 送入行地址寄存器，经译码驱动形成 4 根行线 x_0 、 x_1 、 x_2 、 x_3 ，分别与各行中各单元的字线相连接；对于某一地址码，有一行被选中，该行字线电平为 0.3 V。低位地址 A_1 、 A_0 送入列地址寄存器，经译码驱动形成 4 根列线 y_0 、 y_1 、 y_2 、 y_3 ，用来选择四组位线；对于某一地址码，有一组位线被选中。

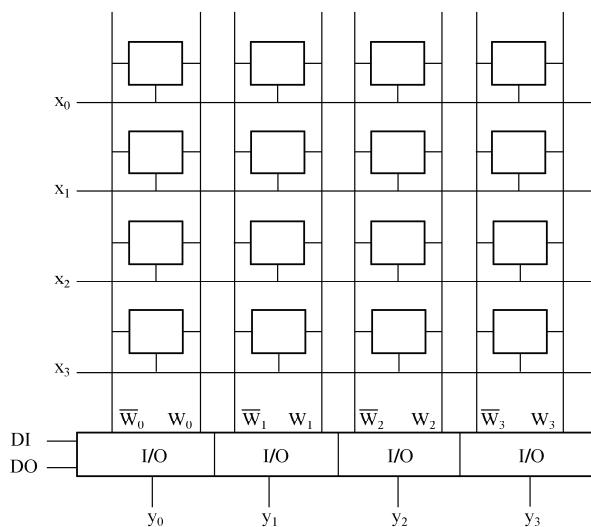
每一组位线有 \overline{W} 、W 两根。写入时，根据这一位的输入 DI，分别决定 \overline{W} 和 W 的电平状态，如图 4-5 所示。读出时，根据在 \overline{W} 或 W 上检测到电流，形成输出信号 0 或 1。相应在位线与数据 I/O 之间，有读写控制电路及读出放大器，在图中简称为 I/O。

【例 4-2】 $\overline{S} = 0$ ， $A_3 \sim A_0 = 0110$ ，则选中 4 个位平面的 x_1 和 y_2 ，即第 1 行与第 2 列交点处的存储单元被选中。若需写入代码 1010，则分别处于不同位平面中的这 4 个被选单元，将分别存入 1010。

以上从原理上描述内部逻辑结构及其寻址过程，实际的电路自然要复杂些。当存储容量扩大时，行列数将随之增加。根据 1993 年资料，商品化双极型存储芯片容量可达 64 KB/片，访问时间在 10 ns 以内。



(a) 4 个位平面的行列译码结构示意图



(b) 一个位平面的行列译码逻辑结构示意图

图 4-5 存储芯片内部译码结构

4.2.2 静态 MOS 存储单元

虽然半导体存储器有多种类型，但其主力则是 MOS（Oxide Semi-conductor）存储器，因为 MOS 工艺存储器芯片的集成度高、功耗小、每位价格低。静态存储器 SRAM 和动态存储器 DRAM 都有非常广泛的应用，而且形成竞争。相比之下，SRAM 的制造工艺比 DRAM 稍复杂，相同体积情况下，每片 SRAM 的容量约为 DRAM 的 1/16。但 SRAM 速率较快，在每片容量相同时，SRAM 的访问时间约为 DRAM 的 1/3~1/2。

1. 静态 MOS 存储单元电路

N 沟道增强型 MOS 存储单元电路，简称 NMOS 六管静态存储单元。六管存储单元的基本结构如图 4-6 所示。

V_1 和 V_3 、 V_2 和 V_4 ，分别是两组 MOS 反相器，其中 V_3 和 V_4 分别是反相器中的负载管。这两个反相器通过彼此交叉反馈，构成一个双稳态触发器。 V_5 和

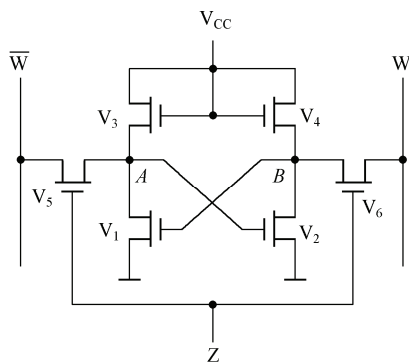


图 4-6 NMOS 六管静态存储单元

V_6 是两个控制门管，由字线控制它们的通断。当字线 Z 加高电平时， V_5 与 V_6 导通，通过一组位线 W 、 \overline{W} ，可对双稳态电路进行读写操作。当字线 Z 为低电平时， V_5 和 V_6 断开，位线脱离，双稳态电路进入信息保持状态。

定义 若 V_1 导通而 V_2 截止，存入信息为 0；若 V_1 截止而 V_2 导通，存入信息为 1。

(1) 写入

字线 Z 加高电平，使控制门管 V_5 与 V_6 导通。

若需写入 0： \overline{W} 加低电平， W 加高电平。 \overline{W} 通过 V_5 使 A 点的结电容放电， A 点变为低电平，使 V_2 截止。而 W 通过 V_6 对 B 点结电容充电至高电平，从而使 V_1 导通。交叉反馈将加快这一状态的变化。

若需写入 1： \overline{W} 加高电平， W 加低电平。 W 通过 V_6 使 B 点结电容放电至低电平，使 V_1 截止。而 \overline{W} 通过 V_5 对 A 点结电容充电至高电平，从而使 V_2 导通。

(2) 保持

字线 Z 加低电平，使 V_5 与 V_6 断开，两根位线 W 和 \overline{W} 与双稳态电路隔离，双稳态电路能依靠自身的交叉反馈保持原有状态不变。

(3) 读出

\overline{W} 和 W 充电至高电平，充电所形成的电平是可浮动的，可随充放电而变。然后对字线 Z 加正脉冲，于是控制门管 V_5 和 V_6 导通。

如果原存信息为 0，即 V_1 导通而 V_2 截止，则字线 Z 加高电平后， \overline{W} 将通过 V_5 和 V_1 对地形成放电回路，因此有电流经 \overline{W} 流入 V_1 ，经放大为“0”信号，表明原存信息为 0。此时因 V_2 截止，所以 W 上无电流通过。

如果原存信息为 1，即 V_1 截止而 V_2 导通，则字线 Z 加高电平后， W 将通过 V_6 和 V_2 对地形成放电回路，因此 W 上将有电流，经放大为“1”信号，表明原存信息为 1。此时因 V_1 截止，所以 \overline{W} 上基本无电流。

总之，位线 \overline{W} 上有电流，为原存信息为 0，位线 W 上有电流，原存信息为 1。上述读出过程并不改变双稳态电路原有状态，因此属于非破坏性读出。

如果将图 4-6 中的负载管 V_3 和 V_4 改用多晶硅电阻代替，则简化为四管静态存储单元电路。四管单元的面积与功耗均只有六管单元的一半，所以集成度能得到很大提高。现在生产的主流 SRAM 多采用四管单元。

2. 静态 MOS 存储芯片举例

Intel 2114 是一种曾广泛使用的小容量 SRAM 芯片，容量为 1 K×4 位。现以它为例说明芯片内部结构、引脚功能及其时序关系。

(1) 内部结构

如图 4-7 所示，2114 内部结构与图 4-5 结构基本相似，但细节上稍有不同。1 K×4=4096，将 4096 个存储单元排成矩阵：64 行×16 列×4 位。6 位地址 $A_3 \sim A_8$ 经过行译码，选中 64 根行线之一。4 位地址 A_0 、 A_1 、 A_2 、 A_9 经过列译码产生 16 根列选择线，每根列选择线同时连接 4 位列线，对应于并行的 4 位，每位列线包含一组位线 \overline{W} 和 W 。因此这种矩阵结构也可理解为 4 个位平面，每个位平面由 64 行×16 列构成，将 16 列×4 视为 64 根列线。

当片选 $\overline{CS}=0$ 且 $\overline{WE}=0$ 时，数据输入三态门打开，列 I/O 电路对被选中的 1 列×4 位（即 64 列中的 4 列）进行写入。当 $\overline{CS}=0$ 且 $\overline{WE}=1$ 时，数据输入三态门关闭，数据输出三态门打开，列 I/O 电路将被选中的 1 列×4 位读出信号送数据线。

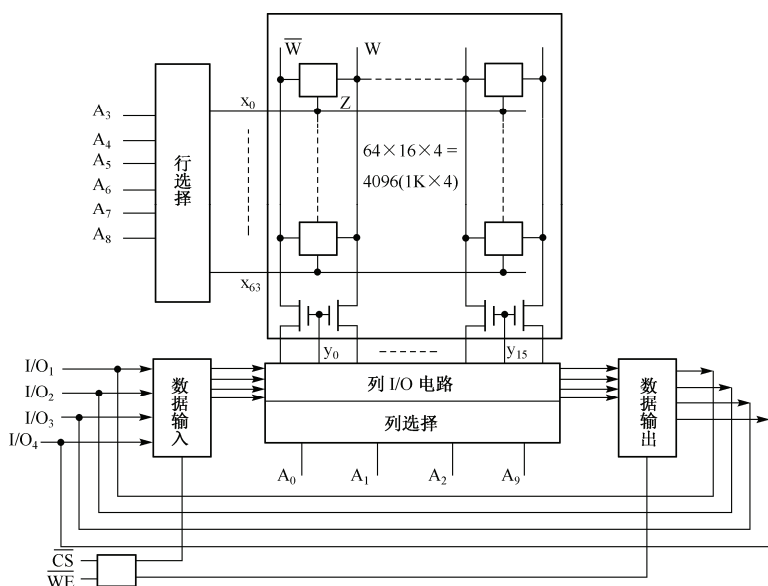


图 4-7 2114 SRAM 芯片内部结构图

(2) 引脚

Intel 2114 是 18 脚封装，如图 4-8 所示。片选 \overline{CS} ，为低电平时选中本芯片；写使能 \overline{WE} ，低电平时写入，高电平时读出；地址 10 位：A₉~A₀，对应于 1K 容量。双向数据线 4 位：I/O₄~I/O₁ 对应于每个编址单元的 4 位。可直接与数据总线连接，输出数据可维持一定时间，以同步送入有关寄存器。当 $\overline{CS}=1$ 时，数据输出呈高阻抗，与数据总线隔离。

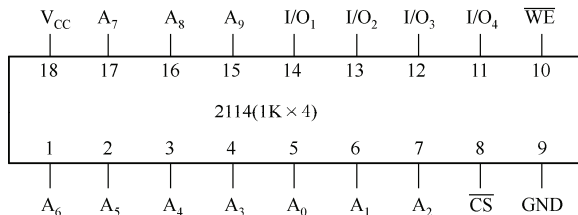


图 4-8 Intel 2114 芯片引脚及功能

(3) 读写时序

为了让芯片正常工作，必须按所要求的时序关系提供地址、数据信息和有关控制信号，如图 4-9 所示。

加到芯片上的地址共 10 位，根据输入的地址编码，有些位为高电平，有些位为低电平，所以图中采取整体示意画法。在片选无效期，数据输出呈高阻抗，图中让 D_{OUT} 位于非高非低的中间位置，以示为浮空状态。数据输入或数据输出有效时，则采取图示的整体示意画法。

① 读周期

在准备好有效地址后，向存储芯片发出片选信号（ $\overline{CS}=0$ ）和读命令（ $\overline{WE}=1$ ），经过一段时间数据输出有效。当读出数据送达目的地后（如读入 CPU），可撤销片选信号与读命令，然后允许更换地址以准备下一个读写周期。有关时间参数如下：

- ⊙ t_{RC} 一读周期。有效地址应当在整个读周期中维持不变， t_{RC} 也是两次读出的最小间隔。
- ⊙ t_A 一读时间，从地址有效到输出稳定所需的时间。此时可以使用读取的数据，但读周期尚未结束，读出时间小于读周期。在数据输出稳定后，允许撤销片选信号和读命令。

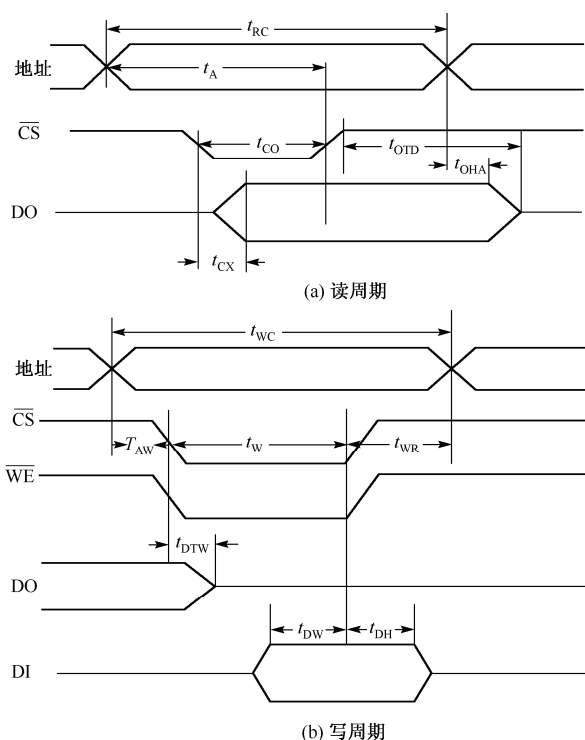


图 4-9 Intel 2114 的读写周期波形

- ⊙ t_{CO} —从片选 \overline{CS} 有效到输出稳定所需的时间。输出稳定后，允许撤销片选和读命令。
- ⊙ t_{CX} —从片选 \overline{CS} 有效到数据有效所需的时间。但此时数据尚未稳定，仅仅是开始出现有效数据而已。
- ⊙ t_{OTD} —从片选信号无效后到数据输出变为高阻抗状态。换句话说， t_{OTD} 是片选无效后输出数据还能维持的时间，在这之后输出将无效。
- ⊙ t_{OHA} —地址改变后数据输出的维持时间。

② 写周期

在准备好有效地址与输入数据后，向存储芯片发出片选信号 ($\overline{CS}=0$) 和写命令 ($\overline{WE}=0$)，经过一段时间，可将有效输出数据写入存储芯片。然后可撤销片选信号和写命令，再经过一段时间，可更换输入数据与地址，准备新的读写周期。有关时间参数如下。

- ⊙ t_{WC} —写周期。在写周期中地址应保持不变， t_{WC} 是两次写入操作之间的最小间隔。
- ⊙ t_{AW} —在地址有效后，经过一段时间 t_{AW} ，才能向芯片发出写命令。如果芯片内地址尚未稳定就发出写命令，有可能产生误写入。
- ⊙ t_W —写时间，即片选和写命令同时有效的时间。 t_W 是写周期的主要时间成分，但小于整个写周期时间。
- ⊙ t_{WR} —写恢复时间。在片选与写命令都撤销后，必须等待 t_{WR} ，才允许改变地址码，进入下一个读写周期。显然，为了保证数据的可靠写入，地址有效时间（写周期时间）至少应满足： $t_{WC} = t_{AW} + t_W + t_{WR}$ 。
- ⊙ t_{DTW} —从写信号有效到数据输出为三态的时间。如图 4-7 所示，当 \overline{WE} 为低后，数据输出门将被封锁，输出呈高阻态，然后才能从双向数据线上输入写数据。 t_{DTW} 是这一转换过程所需的时间。

- ④ t_{DW} —数据有效时间。从输入数据稳定到允许撤销写命令和片选，数据至少应维持 t_{DW} 时间，方能保证可靠写入。
- ⑤ t_{DH} —写信号撤销后的数据保持时间。

对于某种存储芯片，上述读写周期时间参数应满足一定的指标要求，可从芯片使用手册查到。相应地，CPU 或其他部件访存时，所发出的有关信号波形应满足这些要求。

3. 静态随机存储器技术

静态存储器最大的优点是访问速率快，常用在速率要求至关重要的应用中。

静态随机存储器 SRAM (Static RAM) 的发展过程主要是提高访问速度和功能多样化。早期 SRAM 的访问速率一般为 300 ns 左右，现在的 SRAM 可以在约 15 ns 的时间内得到要访问的数据。特别是支持突发操作的同步 SRAM 推出之后，作为现代微型计算机的第二级 Cache，不需插入等待时钟（即“零等待”），即可与现在最快的微处理器匹配使用。SRAM 的另一个发展特点就是功能的多元化，为了适应实际应用的需要，开发出了众多的产品，如支持缓冲操作的先进先出存储器 FIFO (First In First Out)、支持数据共享的多端口 SRAM、掉电时信息不丢失的非挥发随机存储器 NV SRAM 和具有高集成度的类静态随机存储器 PSRAM。

FIFO 是一种顺序存储的静态存储器，在内部结构上与随机存储器存在较大的区别，主要用在需要进行数据缓冲的地方，如不同传输速率总线之间的接口电路中。

多端口 SRAM 是一种具有多个数据访问端口的静态随机存储器，多个目标设备可以同时访问这种类型的存储器，以实现数据的共享，主要用在需要进行数据高速共享的场合，如对某一数据源要实时监控（送到显示缓冲区进行显示）和处理（如进行压缩存盘或送到通信信道）时，可能需要采用这种多端口的 SRAM。

典型的 NV SRAM 由低功耗的 SRAM、能够测量电压的存储器控制器和一个锂电池共同组成，当供给存储芯片的电源电压低于维持 SRAM 操作的电压时，芯片中的存储器控制器将供电切换到内部的锂电上。这样，NV SRAM 就可以保证存储在 SRAM 中的内容不丢失，成为一种非挥发的静态存储器。NV SRAM 存储器主要用于需要进行高速操作的永久信息保存场合，如高速调制解调器中，而前面提到的各种只读存储器由于其访问速度慢而无法替代 NV SRAM。

4.2.3 动态 MOS 存储单元

PSRAM 虽然对外表现出静态随机存储器 (SRAM) 的特性，但实际上它却是一种典型的动态随机存储器 (DRAM)，存储芯片内部也集成了动态刷新逻辑。PSRAM 同时兼具动态随机存储器集成度高和静态随机存储器接口简单的优点，但由于它的接口电路存在局限（过于复杂），使其只能在一些特殊场合中使用。动态 MOS 存储器的基本存储原理是：将存储信息以电荷形式存于电容之中，这种电容可以是 MOS 管栅极电容或者专用的 MOS 电容，通常将其定义为电容充电至高电平，对应的信息为 1；放电至低电平，则对应的信息为 0。

用电容存储电荷的方式来存储信息，不需要双稳态电路，因而可以简化结构。充电后 MOS 管断开，既可使电容电荷的泄放极少，还能大大降低芯片的功耗。这两点都使芯片集成度得到提高，在相同体积的半导体工艺条件下，DRAM 的最大容量约为 SRAM 的 16 倍。

但是在 MOS 管断开之后，难以使泄漏电阻达到无穷大，电容总会存在泄漏通路。时间过长，电容上的电荷会通过泄漏电阻放电，导致所存储的信息丢失。为此，经过一定时间后就需要对存储内容重写一遍，也就是对存储 1 的电容重新充电，称为刷新。由于这种存储器需要定期刷新才能保持信息不变，所以称为动态存储器。用这种存储机制实现的随机读写存储器就简称为动态存

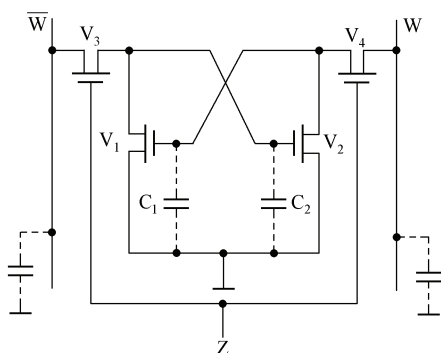


图 4-10 动态 MOS 四管存储单元

储器 (Dynamic RAM, DRAM)。

早期的动态 MOS 存储单元是从静态六管单元简化而来的, 称为四管单元, 后来又简化为三管单元, 再进一步简化为单管单元。本节仅简单介绍四管单元和单管单元两种。

1. 动态 MOS 四管存储单元举例

图 4-10 是动态 MOS 四管存储单元的电路结构。依靠 V_1 和 V_2 的栅极电容存储电荷来存储信息。若 C_1 充电至高电平使 V_1 导通, 而 C_2 放电至低电平使 V_2 截止, 存入信息为 0; 若 C_1 放电至低电平使 V_1 截止, 而 C_2 充电至高电平使 V_2 导通, 则存入信息为 1。

控制门管 V_3 和 V_4 由字线 Z 控制其通断。读写时, 字线 Z 加高电平, V_3 与 V_4 导通, 使存储单元与位线 \overline{W} 、 W 连接。保持信息时, 字线 Z 加低电平, V_3 与 V_4 断开, 使位线与存储单元隔离, 依靠 C_1 或 C_2 存储电荷暂存信息。刷新时, V_3 与 V_4 需导通。

注意: 与六管静态存储单元结构相比, 四管动态单元中没有负载管 V_5 和 V_6 。 V_3 与 T_4 断开后, V_1 与 V_2 之间并无交叉反馈, 因此四管单元并非双稳态电路。

① 写入: 字线 Z 先加高电平, 使 V_3 与 V_4 导通。

若要写入 0, 则 \overline{W} 加低电平, W 加高电平, W 通过 V_4 对 C_1 充电至高电平, 使 V_1 导通。而 C_2 通过两个放电回路放电, 一条是通过 V_1 放电, 另一条是通过 V_3 对 \overline{W} 放电, C_2 放电至低电平, 使 V_2 截止。

若要写入 1, 则 \overline{W} 加高电平, W 加低电平。 \overline{W} 通过 V_3 对 C_2 充电至高电平, 使 V_2 导通。而 C_1 通过两个放电回路放电, 一条是 C_1 通过 V_2 放电, 另一条是 C_1 通过 V_4 对 W 放电, 会使 C_1 放电至低电平, 导致 V_1 截止。

② 暂存信息: 字线 Z 加低电平, V_3 与 V_4 断开, 基本上无放电回路, 内部仅存在电容上发生的泄漏电流, 信息可暂存数毫秒。

③ 读出: 先对位线 (读出线) \overline{W} 和 W 预充电, 也就是对位线的分布电容充电至高电平, 然后断开充电回路, 使 \overline{W} 和 W 处于可浮动状态。再对字线 Z 加高电平, 使 V_3 与 V_4 导通, 此时 \overline{W} 与 W 成为读出线。

若原存信息为 0, 即 C_1 上有电荷为高电平, V_1 导通而 V_2 截止。 \overline{W} 则通过 V_3 与 V_1 对地放电, \overline{W} 电平下降, \overline{W} 上将有放电电流流过, 放大后作为 0 信号, 简称为读出 0。与此同时, W 通过 V_4 对 C_1 充电, 可补充泄漏掉的电荷。由此可见, 四管单元为非破坏性读出, 且读出的过程可起刷新 (也可视为回写) 作用。

若原存信息为 1, 即 C_2 上有电荷为高电平, V_1 截止而 V_2 导通。 W 则通过 V_4 与 V_2 对地放电, W 上有放电电流流过, 放大后作为 1 信号, 简称读出 1。同时, \overline{W} 通过 V_3 对 C_2 充电, 补充电荷。

四管动态存储单元仍保持互补对称结构, 读写操作比较可靠, 外围电路较简单, 读出过程可起到对存储电路进行刷新的作用。但每个存储单元所用元件仍嫌过多, 使每片容量受限。当每片容量在 4 KB 以下时, 可采用四管单元。

2. 单管动态存储单元

为了进一步简化结构, 单管动态存储单元中只有一个电容和一个 MOS 管, 如图 4-11 所示。

电容 C 用来存储电荷, 控制管 V 用来控制读写。读写时, 字线 Z 加高电平, V 导通。暂存信息时, 字线加低电平使 V 断开, C 基本上无放电回路 (当然还有一定的泄漏)。当电容 C 上充电到高电平时, 存入信息为 1; 当电容 C 放电到低电平时, 存入的信息为 0。

① 写入: 字线 Z 加高电平, 使 V 导通。若要写入 0, 则 W 线加低电平, 电容 C 通过控制管 V 对 W 放电, 呈低电平 V_0 ; 若要写入 1, 则 W 线加高电平, W 通过 V 对电容 C 充电, C 上有电荷呈高电平 V_1 。

② 暂存信息: 字线 Z 加低电平, 使 V 断开, 电容 C 基本上没有放电回路, 其上的电荷 (信息) 可暂存数毫秒, 或者维持无电荷的 0 状态。

③ 读出: 先对位线 (读出线) W 预充电, 使其分布电容 C' 充电至 V_m , 其浮动值等于 $V_m = (V_1 + V_2)/2$; 然后对字线 Z 加高电平, V 导通。

如果原存信息为 0, 则 W 将通过 V 向电容 C 充电, W 本身的电平将下降, 按 C 与分布电容 C' 的电容值决定新的电平值。

如果原存信息为 1, 则电容 C 将通过 V 向位线 W 放电, 使 W 电平上升。

根据 W 线电平变化的方向及幅度, 可鉴别原存信息是 0 还是 1。显然, 读操作后 C 上的电荷将发生变化, 属于破坏性读出, 需要读后重写 (再生)。这一过程由芯片内的外围电路自动实现, 使用者不必关心。

与四管单元相比, 单管存储单元将结构简化到了最低程度, 因而集成度高, 但要求的读写外围电路 (片内) 较复杂一些。当每片容量在 4 KB 以上时, 多采用单管存储单元。

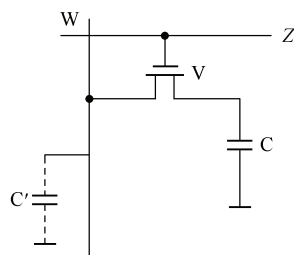


图 4-11 单管动态存储单元

3. DRAM 芯片举例

Intel 2164 是一种 DRAM 芯片, 每片容量 $64K \times 1$ 位。早期的微机中曾用这种芯片构成主存储器。现以它为例, 说明 DRAM 芯片的内部结构、引脚功能及其读/写时序。

(1) 内部结构

2164 芯片容量是 $64K \times 1$ 位, 本应构成一个 256×256 的矩阵, 为提高工作速度 (需减少行列线上的分布电容), 在芯片内部分为 4 个 128×128 矩阵, 每个译码矩阵配有 128 个读出放大器, 各有一套 I/O 控制 (读写控制) 电路, 如图 4-12 所示。

$64K$ 个编址需要 16 位地址寻址, 但芯片引脚只有 8 根地址线 $A_7 \sim A_0$, 需分时复用。先送入 8 位行地址, 在行选信号 \overline{RAS} 控制下送入行地址锁存器, 锁存器提供 8 位行地址 $RA_7 \sim RA_0$, 译码后产生 2 组行选择线, 每组 128 根。然后送入 8 位列地址, 在列选信号 \overline{CAS} 控制下送入列地址锁存器, 锁存器提供 8 位列地址 $CA_7 \sim CA_0$, 译码后产生 2 组列选择线, 每组也是 128 根。行地址 RA_7 与列地址 CA_7 选择 4 套 I/O 控制电路之一与四个译码矩阵之一。因此, 16 位地址是分成两次送入芯片的, 对于某一地址码, 只有一个 128×128 矩阵及其 I/O 控制电路被选中。

(2) 引脚

Intel 2164 是 16 脚封装, 如图 4-13 所示。

- ⊙ 地址 8 位: $A_7 \sim A_0$, 兼作行地址与列地址, 分时复用。
- ⊙ 行选 \overline{RAS} , 低电平时将 $A_7 \sim A_0$ 作为行地址, 送入芯片内的行地址锁存器。
- ⊙ 列选 \overline{CAS} , 低电平时将 $A_7 \sim A_0$ 作为列地址, 送入芯片内的列地址锁存器。可见, 片选信号已分解为行选与列选两部分。
- ⊙ 数据输入 DI。

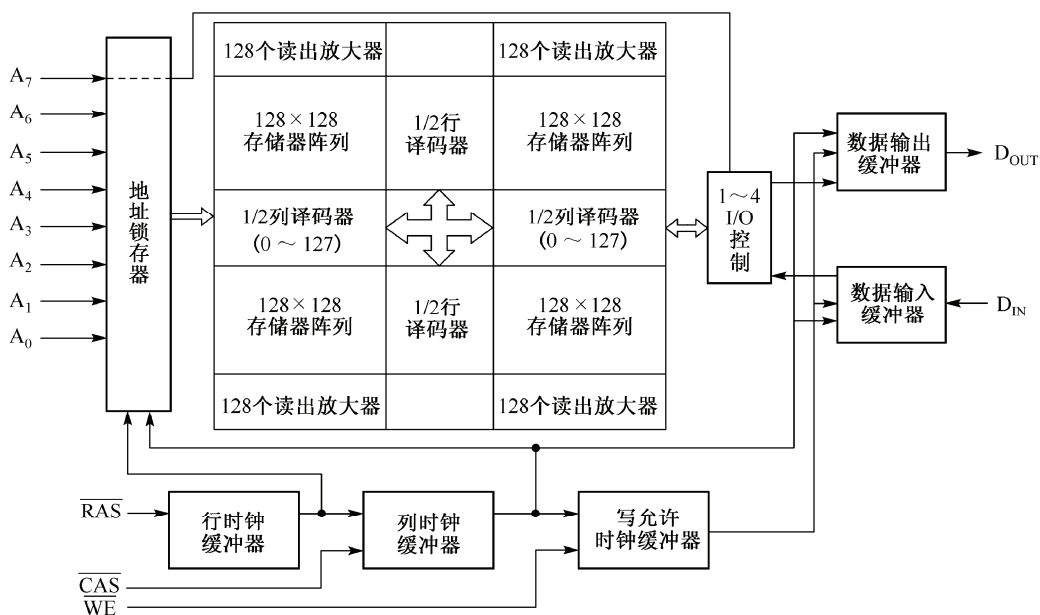


图 4-12 Intel 2164 DRAM 芯片内部结构图

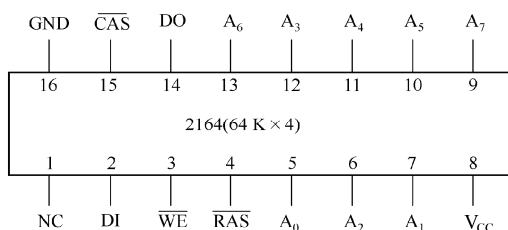


图 4-13 Intel 2164 芯片引脚及功能

⊙ 数据输出 DO。

⊙ \overline{WE} ，低电平 ($\overline{WE}=0$) 时写入，高电平 ($\overline{WE}=1$) 时读出。

引脚 1 空闲未用，在该 DRAM 系列的新产品中，将引脚 1 作为自动刷新端。将行选信号送到引脚 1，可在芯片内自动实现动态刷新。

(3) 读/写时序

① 读周期

在准备好行地址后，发行选 ($\overline{RAS}=0$) 将行地址打入片内的行地址锁存器。为使行地址可靠输入，发出行选后，行地址需维持一段时间才能切换。

如果在发列选之前先发读命令，即 $\overline{WE}=1$ ，将有助于提高读出速度。

在准备好列地址后，发列选 ($\overline{CAS}=0$)，此时行选不撤销。在发出列选后，列地址应维持一段时间，以打入列地址锁存器。此后允许更换地址，为下一个读写周期做准备。

读周期的主要时间参数如下：

⊙ t_{RC} — 读周期时间，即两次发出行选信号之间的时间间隔。

⊙ t_{RP} — 行选信号恢复时间。

⊙ t_{RAC} — 从发出行选信号到数据输出有效的的时间。

⊙ t_{CAC} — 从发出列选信号到数据输出有效的的时间。

⊙ t_{RO} — 从发出行选信号到数据输出稳定的时间。

② 写周期

在准备好行地址后，发行选（ $\overline{\text{RAS}}=0$ ），此后行地址需要维持一段时间，才能切换为列地址。如图 4-14 所示，虽然发出了写命令（ $\overline{\text{WE}}=0$ ），但在发列选信号之前没有列线被选中，因而还未真正写入，只是开始写的准备工作。在准备好列地址并输入数据后，才能发列选信号（ $\overline{\text{CAS}}=0$ ）。此后，列地址和输入数据均需维持一段时间，待列地址打入列地址锁存器后，方可撤换列地址。待可靠写入之后，才能撤销输入数据。

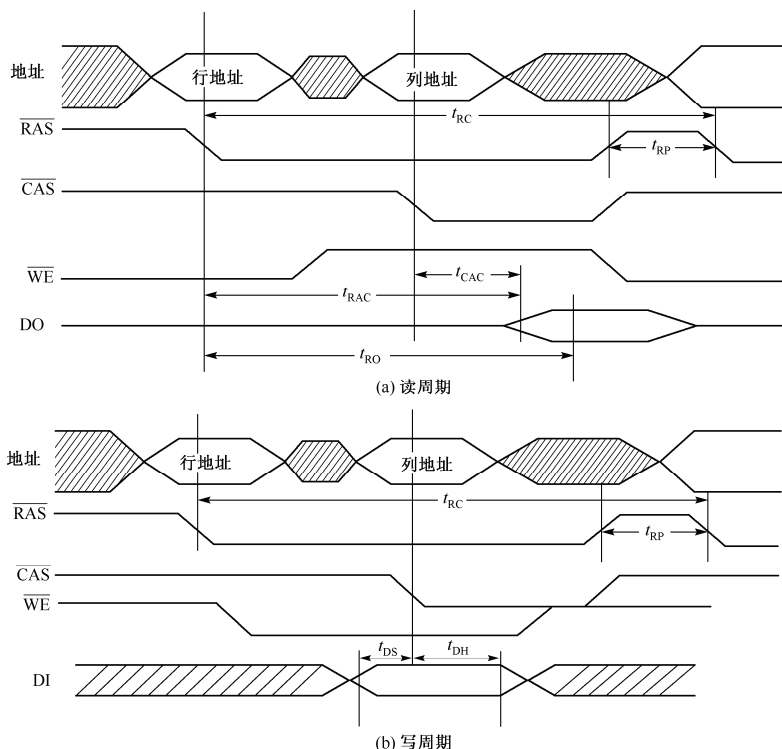


图 4-14 Intel 2164 的读写周期波形图

写周期的主要时间参数如下：

- ⊙ t_{RC} —写周期时间。在实际系统中读写周期时间安排得相同，所以 t_{RC} 又称为存取周期或读写周期。
- ⊙ t_{RP} —行选信号恢复时间，因此行选信号宽度为 $t_{\text{RC}} - t_{\text{RP}}$ 。
- ⊙ t_{DS} —从数据输入有效到列选和写命令均有效，即写入数据建立时间。
- ⊙ t_{DH} —当写命令与列选信号均有效后，数据保持的时间。

4. 其他 DRAM 芯片简介

(1) 41128 芯片

如图 4-15 所示，41128 是 $128\text{K} \times 1$ 位的 DRAM 芯片，它将 $128\text{K} \times 1$ 分为两个 $64\text{K} \times 1$ 模块。地址引脚仍只有 8 位，分时复用作为行地址与列地址，但行选信号分为两个： $\overline{\text{RAS}}_0$ 和 $\overline{\text{RAS}}_1$ 。对于某个地址编码，只有一个行选信号有效，选中芯片中的一个模块，再由 8 位行地址和 8 位列地址选中该模块中的某个单元。这提供了又一种扩大芯片容量的方法，即增加行（列）选信号，而保持地址位数不变。

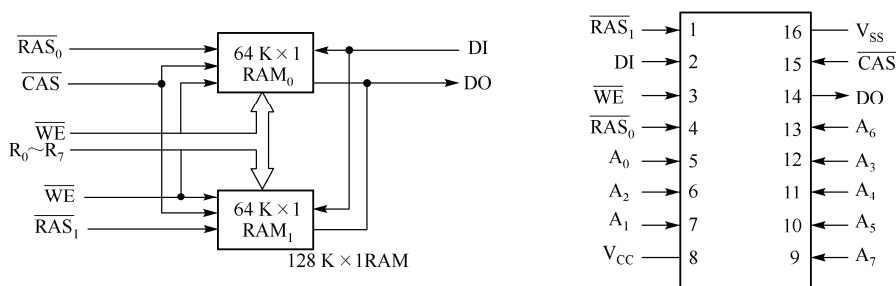


图 4-15 41128 DRAM 芯片

(2) iRAM 芯片

iRAM (Integrated RAM) 即集成化 RAM, 如图 4-16 所示, 它将一个 DRAM 系统集成在一块芯片内, 包括存储矩阵、行列译码与读写控制、地址分时输入、数据 I/O 缓冲器、控制逻辑及时序发生器、刷新逻辑、访存/刷新裁决逻辑等。

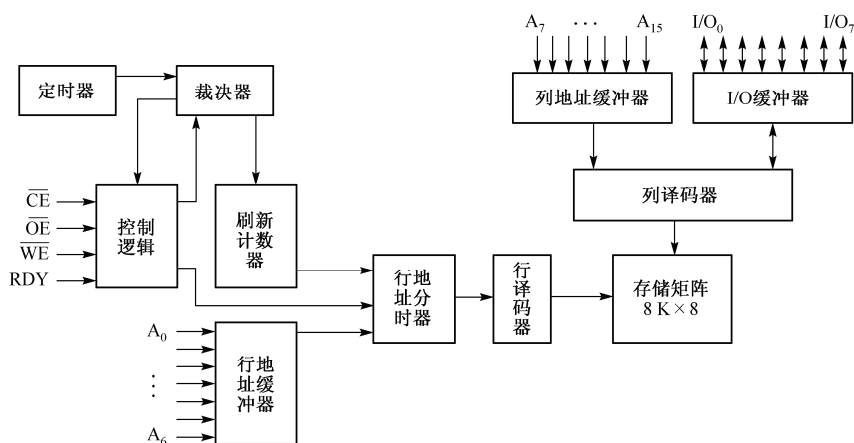


图 4-16 iRAM 芯片的逻辑组成

由于这种存储器采用的单管动态存储单元, 因此其芯片内部一般都应有专门进行存储单元动态刷新的相关电路逻辑。在进行刷新时, 可以先由刷新计数器产生刷新单元的行地址, 并由裁决器决定是接收地址进行读写, 还是内部刷新。从使用者的角度看, 这种芯片的使用特性与静态 RAM 相同, 不再需要外部刷新电路, 因此被称为准静态 RAM, 它兼有静态 RAM 使用方便及动态 RAM 密度高、功耗低的优点。

4.2.4 半导体只读存储器

随着微电子技术的高度发展, 半导体只读存储器技术也得到了长足的进步。从只读存储器 ROM 来看, 最早推出的是掩膜型只读存储器 MROM, 这种 ROM 只能由生产厂家写入存储信息, 因此, 对要进行硬件开发的用户极不方便。接着出现的是用户可一次性写入的只读存储器 PROM。随后, 又推出了用户可多次修改的 EPROM 只读存储器, 这种存储器在用户写入信息后若要修改, 需使用特殊的设备用紫外线对 EPROM 芯片长时间直接照射, 才能擦除掉所保存的信息, 然后在专用的编程器上加入高电压的编程脉冲进行信息的再次写入。这种器件曾被广泛地应用过一段时间, 但对硬件开发者来说, 仍然不是很方便。随后出现了电可擦除的 E2PROM 只读存储器, 这种存储器无需紫外线长时间照射进行信息的擦除, 而是用加反向电压的方式进行原信息的擦除,

并且可以按存储位进行擦除。这种方式比整个芯片的擦除和信息修改速度快。现在推出的新一代只读存储器是 FLASH，称为闪存，由于这种存储器可以做到在线改写，即不需使 FLASH 离开系统即可修改其中的内容，这在过去的只读存储器是做不到的，因此使用 FLASH 比使用 E2PROM 更灵活和更方便。并且，FLASH 存储器具有更高的集成度和更低的功耗，因此现在被广泛使用。只读存储器的发展主要是方便用户的使用过程。以下对几种主要的只读存储器和芯片作介绍。

(1) 掩膜型只读存储器 MROM (Mask-ROM)

在制造 MROM 芯片之前，先由用户提供所需存储的信息，以 0、1 代码表示。芯片制造厂据此设计相应的光刻掩模，以有无元件表示 1、0。因此这种芯片中的信息是固定不变的，使用时只能读出而不能写入新内容，即不能改写，适用于需要量大且不需改写的场合。例如，显示器和打印机中的字符发生器，根据字符编码输出字符形状的点阵代码；又如，固定不变的微程序代码；再如，代码转换一类的输入/输出转换逻辑等。

(2) 可程序（一次编程型）只读存储器 PROM (Programmable-ROM)

芯片出厂时各单元内容全为 0，用户可用专门的 PROM 写入器将信息写入，所以称为可编程型。这种写入是不可逆的，某单元一旦写入 1，就不能再次将其改写为 0，即只能执行一次写入操作，因此也可将其称为一次编程型。

有一种写入原理属于结破坏型的，即在行列线交点处制作一对彼此反向的二极管，它们由于反向而不能导通，称为 0。若该位需要写入 1，则在相应行列线之间加较高电压，将反偏的一只二极管永久性击穿，留下正向可导通的一只二极管，称为写入 1，显然这也是不可逆的。

更常用的一种写入原理属于熔丝型，制造时在行列交点处连接一段熔丝，即易熔材料，称为存入 0。若该位需写入 1，则让它通过较大电流，使熔丝熔断，显然这也是不可逆的。

图 4-17 是一个写入了信息的 PROM 内部逻辑结构图。外部的地址输入经行译码选择某一行线（字线），即某一存储单元（字）。通过列线输出读出信息，即各位输出。图示的 PROM 是 4×4 容量，0^o单元到 3^o单元，所存信息分别是 0110、1011、1010、0101。用户可购得通用的 PROM 芯片，写入前内容为全 0。然后根据需要，写入不再变化的内容，如可固化的程序、微程序、标准字库、代码转换表等。

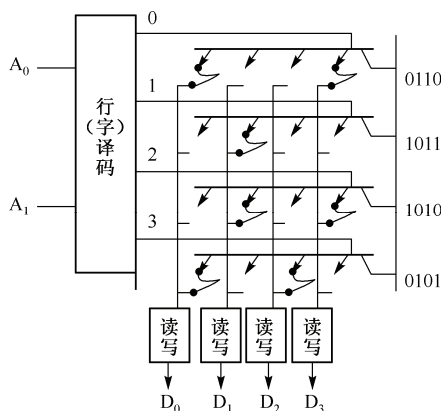


图 4-17 熔丝型 PROM 原理图

PROM 片内的行译码器实际上是一个固定的、不可编程设置的与门阵列，存储体则是一个可一次编程设置的或门阵列。众所周知，用卡诺图等工具化简后的组合逻辑函数，其基本形态就是与-或项。因此也可用 PROM 产生多输入变量与多输出的组合逻辑函数，减少逻辑电路元件数，从而简化电路板结构。也有人将广泛应用的与或门阵列器件，（如可程序逻辑阵列 PLA、可程序阵列逻辑 PAL 等）视为一种特殊的只读存储器。

(3) 可擦除重编程只读存储器 EPROM (Erasably-Programmable ROM)

EPROM 可用专门的写入器在 +25 V 高压下写入信息，在 +5 V 的正常电压下只能读出不能写入，用紫外线照射一定时间后可擦除原存信息，然后重新写入，因此被称为可重编程（即可改写）的只读存储器。可擦除、可重写是在特殊环境下，在工作环境中，EPROM 则是只读不写的存储器。这对用户的应用显然更方便，因而应用非常广泛。但它的可重写次数是有限的，目前的产品

只允许重写几十次，甚至更低。

图 4-18 是存储一位信息的浮栅型 P 沟道 MOS 管结构示意图。在 N 型硅衬底上制造了两个 P+区，分别引出源极 S 和漏极 D。在 S 与 D 之间，有一个用多晶硅做成的栅极，它被埋在氧化硅绝缘层之中，不与外部通导，因而称为浮栅。在芯片制成后，写入信息前，浮栅上没有电荷，两个 P+区之间没有导通沟道，因此在+5 V 电源下 S 与 D 之间不通导，一般定义为 1。所以，写入前芯片内容为全 1。

如果在 S 与 D 之间加+25 V 高压（S 正，D 负），由于浮栅与硅基片之间的绝缘层很薄，只有 0.05~0.1 μm ，于是在 D 极附近的强电场作用下，D 与浮栅之间被瞬时击穿，大量电子注入浮栅。当 25 V 高压撤除后，绝缘层恢复绝缘状态，浮栅上电子的能量不足以使电子穿越绝缘层。如果不外加能量，浮栅上的电子可以长期保留。由于浮栅上带负电荷，在硅基片的对应一边将形成带正电荷的 P 沟道，如图 4-18 所示状态。如果在 S 与 D 间加工作电压，MOS 管将呈导通状态，定义为 0。所以，对 EPROM 写入的过程是将有关位单元由 1 改写为 0，写入内容可长期保持不变，因而在+5 V 电源下只读不写。

芯片封装上方装有一个石英玻璃窗口，这是 EPROM 芯片的外形特征。当用紫外线照射时（典型方式是用 12 mW/cm^2 功率的紫外线灯，照射 10~20 分钟），浮栅上的电子获得能量，将能越过绝缘层泄放掉。浮栅失去电荷，P 沟道消失，芯片被擦除为全 1。显然，写入过程可选择字、位逐个地写入，而紫外线擦除是将全芯片擦除为 1。当重新击穿写 0、照射擦除的过程反复进行了一定次数后，绝缘层将被永久性地击穿，芯片损坏。因此应当尽量减少重写次数。此外，在阳光或荧光灯照射下过长（一周以上），信息也会被丢失。所以，要注意 EPROM 芯片的使用环境，例如用保护膜遮盖窗口，当需要擦除时再打开。

图 4-19 所示的 Intel 2716 是一种常用的 EPROM 芯片，容量为 2K×8 位。现以它为例说明 EPROM 的工作方式，如表 4-1 所示。

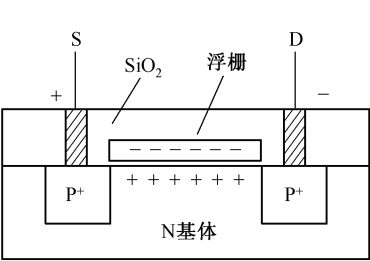


图 4-18 P 沟道 MOS 管结构示意图

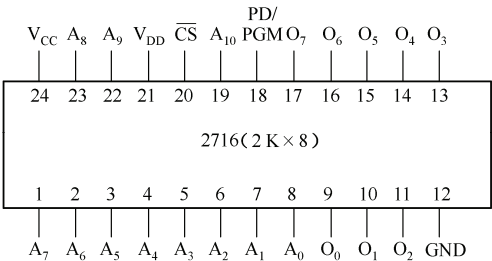


图 4-19 Intel 2716 EPROM 芯片引脚及功能

表 4-1 Intel 2716 芯片的工作方式选择

| 工作方式 | V_{CC} | V_{PP} | \overline{CS} | $O_0 \sim O_7$ | PD/PGM |
|------|----------|----------|-----------------|----------------|-----------|
| 编程写入 | +5 V | +25 V | 高 | 输入 | 50 ms 正脉冲 |
| 读 | +5 V | +5 V | 低 | 输出 | 低 |
| 未选中 | +5 V | +5 V | 高 | 高阻 | 无关 |
| 功耗下降 | +5 V | +5 V | 无关 | 高阻 | 高 |
| 程序验证 | +5 V | +25 V | 低 | 输出 | 低 |
| 禁止编程 | +5 V | +25 V | 高 | 高阻 | 低 |

① 编程写入。将 EPROM 芯片置于专门的写入器中，由 V_{PP} 端引入+25 V 高压， \overline{CS} 为高， $A_{10} \sim A_0$ 选择写入单元， $O_0 \sim O_7$ 输入待写信息，编程端 PGM 引入一个正脉冲，其宽度为 45~55 ms，幅度为 TTL 高电平，则按字节写入 8 位信息。PGM 正脉冲过窄则不能可靠写入，若过宽则可能

损伤芯片。有多种形式的 EPROM 写入器，常见的写入方式是将写入器与微机系统相连，以微机为宿主机，在宿主机上可编程连续对芯片各单元依次写入。

② 读。芯片写入后插入存储系统，只引入+5 V 电源，PD/PGM 端为低。若片选有效，则按地址读出，由 $O_0 \sim O_7$ 输出到数据总线。这是 EPROM 在正常工作时的方式，只读不写。

③ 未选中。若片选为高，即未选中本芯片，输出呈高阻，不影响数据总线状态。

④ 功耗下降。虽然芯片处于+5 V 的正常电源之下，但不要求它工作，为降低功耗，可使芯片处于一种低功耗的备用状态。PD/PGM 是功耗下降/编程（Power Down/Program）端，若 PD 端为高，但 V_{PP} 为+5 V（不能写入），则芯片输出呈高阻，且功耗只有原来的 1/4，如 2716 芯片功耗由 525 mW 下降到 132 mW。

⑤ 程序验证。芯片位于写入器环境 $V_{PP}=+25\text{ V}$ ，但已编程写入完毕。可让片选 \overline{CS} 有效，但编程控制端 PGM 为低（非写入状态），此时 $O_0 \sim O_7$ 可以按地址输出已经写入的内容，供验证写入是否正确。若无误，可将芯片取出，插入使用该芯片的系统。

⑥ 禁止编程。芯片位于写入环境， $V_{PP}=+25\text{ V}$ ，但 PGM 为低， \overline{CS} 为高，则 $O_0 \sim O_7$ 呈高阻与数据线脱离，禁止读写数据。

以上 6 种状态中，①、⑤、⑥属于写入器环境，其余 3 种属于应用环境。

（4）电可擦除重编程只读存储器 EEPROM（Electrically EPROM）

常规的 EPROM 芯片需用紫外线照射才能擦除，仍嫌不方便。随着存储芯片制造技术的进展，出现了可加高压擦除的只读存储器，即电可改写（重编程），缩写为 EEPROM 或 E2PROM。EEPROM 采用金属-氮-氧化硅（NMOS）集成工艺，仍可实现正常工作方式中的只读不写，但在擦除时只需加高压对指定单元产生电流，形成“电子隧道”，将该单元信息擦除，而其他未通电流的单元内容保持不变。显然，E2PROM 比 EPROM 更方便，但它仍需在专用的写入器中进行擦除和改写。

（5）新一代可编程只读存储器 FLASH

20 世纪 80 年代中期研制出一种快速擦写型存储器（FLASH Memory），具备 RAM（随机存储器）和 ROM（只读存储器）的所有功能，而且功耗低、集成度高，发展前景非常广阔。这种器件沿用了 EPROM 的简单结构和浮栅/热电子注入的编程写入方式，又兼备 E2PROM 的可电擦除特点，而且可在计算机内进行擦除和编程写入，因此称为快擦写型电可重编程，即 FLASH E2PROM。

FLASH 通常也称为“闪存”，具有掉电时信息不丢失、块擦除、单一供电、高密度的信息存储等显著特点，从而得到广泛应用，如 U 盘等，主要用于保存系统引导程序和系统参数数据等需要长期保存的各种信息。FLASH 的典型结构和逻辑符号如图 4-20 所示。

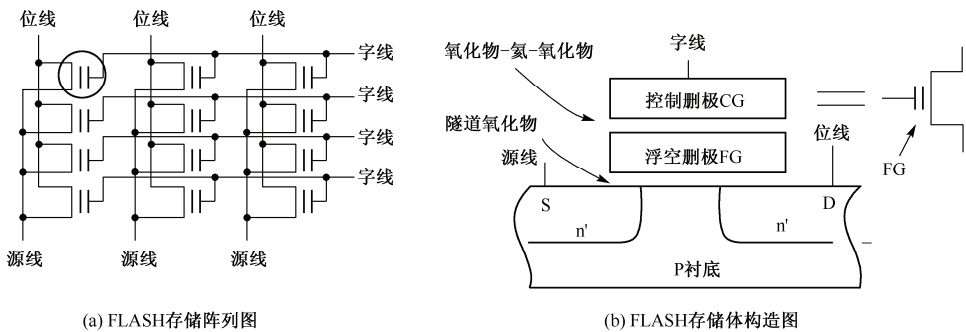


图 4-20 FLASH 的典型结构和逻辑符号

FLASH 的信息存储电路由一个晶体管构成,通过沉积在衬底上被氧化物包围的多晶硅浮空栅来保存电荷,以此维持衬底上源、漏极之间导电沟道的存在,从而保持其上的信息存储。若浮空栅上保存有电荷,则在源、漏极之间形成导电沟道,为一种稳定状态,可以认为该单元电路保存“0”信息;若浮空栅上没有电荷存在,则在源、漏极之间无法形成导电沟道,为另一种稳定状态,可以认为该单元电路保存“1”信息。

上述这两种稳定状态可以相互转换:状态“0”到状态“1”的转换过程,是将浮空栅上的电荷移走的过程。在栅极与源极之间加一个正向电压 V_{sg} ,在漏极与源极之间加一个正向电压 V_{sd} ,保证 $V_{sg} > V_{sd}$,来自源极的电荷向浮空栅扩散,使浮空栅上带上电荷,在源、漏极之间形成导电沟道,完成状态的转换,该转换过程称为对 FLASH 编程。进行正常的读取操作时只要撤销 V_{sg} ,加一个适当的 V_{sd} 即可。据测定,正常使用情况下在浮空栅上编程的电荷可以保存 100 年而不丢失。

由于 FLASH 只需单个器件(即一个晶体管)即可保存信息,因此具有很高的集成度,这与单管 DRAM 类似,访问速度也几乎接近 EDO 类型的 DRAM。供电撤销之后,保存在 FLASH 中的信息不丢失,FLASH 又具有只读存储器的特点;对其擦除和编程时,只要在源、栅极或漏、源极之间加一个适当的正向电压即可,可以在线擦除与编程,FLASH 又具有 E^2 PROM 的特点;对 FLASH 进行擦除时是按块进行的,这又具有 EPROM 的整块擦除的特点。总之,FLASH 是一种高集成度、低成本、高速、能够灵活使用的新一代只读存储器。

在现阶段,FLASH E^2 PROM 除常被用来取代 EPROM 和 E^2 PROM 外,还进一步被用来部分地取代磁盘存储器。因为这种芯片具有非易失性,当电源断开后仍能长久保存信息,属于非易失性半导体存储器,也不需后备电源。从速度上讲,它的读取速度与 DRAM 芯片相近,是磁盘读取速度的 100 倍左右,而它的写数据时间(快擦写)则与硬盘相近,因此它适于做成半导体硬盘,即用半导体存储器构成、当成普通磁盘来调用,这就是目前已逐渐应用的 SSD (Solid State Disk) 即固态硬盘。由于 SSD 没有机电运动,所以比传统磁盘的可靠性更高,它对传统磁盘形成了一种有力挑战。在某些应用中,FLASH E^2 PROM 甚至可以部分取代 DRAM 和 SRAM。

4.3 主存储器的组织

从计算机组成原理的角度看,我们更关心如何用存储芯片组成一个实际的存储器。现在,主存储器是用半导体存储器构成的,其中可能使用 SRAM 芯片,也可能选用 DRAM 芯片。如果主存中有固化区,就需部分使用只读存储器如 EPROM 等。主存储器的组织涉及如下一些问题:

- ◎ 存储器基本逻辑设计,而半导体存储器的逻辑主要是寻址逻辑,即如何按地址选择芯片与片内存储单元。
- ◎ 如果采用 DRAM,还需考虑存储器的动态刷新问题。
- ◎ 所构成的主存如何与 CPU 连接、匹配。
- ◎ 主存校验,如何保证存取信息的正确性。

4.3.1 主存的设计原则

在设计和组成计算机系统的主存储器时,往往需要选择一种或几种存储器芯片构成主存系统,并通过总线把 RAM、ROM 芯片与 CPU 连接起来,并使之协调工作。CPU 对主存储器进行读、写时,总是先输出地址,然后送出读/写命令,最后才能通过数据总线进行信息交换。所以 CPU 与存储器之间连接,必须考虑信号线的连接、时序配合、驱动能力等问题。

存储器与 CPU（或总线控制器）的连接主要由三部分组成：地址总线的连接、数据总线的连接和控制总线的连接。

（1）驱动能力

在与总线连接时，先要考虑 CPU（或总线控制器）的驱动能力方面的问题。对于 CPU（或总线控制器），一般输出线的直流负载能力都是很有限的，尽管可能经过了驱动放大，且现代存储器都是直流负载很小的 CMOS 或 CHMOS 电路，但由于分布于总线和存储器上的负载电容总是存在的，所以要保证所设计的存储系统稳定工作，就必须考虑输出端能带负载的最大能力。若负载太重，则必须增加信号的缓冲驱动。

（2）存储器芯片类型选择

根据主存储器各区域的应用不同，在构成主存储器系统时，应选择适当的存储器芯片。如前所述，半导体存储器可分成 RAM 和 ROM 两大类。RAM 最大的特点是其存储的信息可以在程序中用读/写指令以随机存取的方式读写。但掉电时信息会丢失，所以 RAM 一般用于存储用户的程序、程序的中间运算结果及掉电时无需保存的 I/O 数据等。

ROM 中的内容掉电时不易失，但不能随机写入，故一般用于存储系统程序、初始化参数和无需在线修改的参数等。其中，掩膜 ROM 和 PROM 用于大批量生产的计算机产品中。当需要多次修改程序或用户自行编程时，宜选用 EPROM 等芯片。E2PROM 多用于保存这样一些数据或参数：它们在系统工作过程中被写入而又需要掉电保护。

（3）存储器芯片与 CPU 的时序配合

存储器的存取时间是反映其工作速度的重要指标，选用存储芯片时，必须考虑它的存取时间和 CPU 的工作速度的匹配问题，即时序配合。

当 CPU 进行读操作时，什么时候送地址信号，什么时候从数据线上读数据，其时序是固定的，而存储器芯片从外部输入地址信号有效，到内部数据送至数据总线上的时序也是固定的，并由存储芯片的内部结构和制造工艺决定。因此把它们连接在一起时，必须注意这两种时序的配合，即当 CPU 发出读数据信号的时候，存储器要把数据输出并稳定在数据总线上，读操作才能顺利进行。

如果存储器芯片读或写周期的工作速度不能满足 CPU 的要求，则可在 CPU 的相应周期内插入一个或数个 T_w 延迟周期，人为延长 CPU 读写时间，使二者相匹配。

为简化外围电路及充分发挥 CPU 的工作速度，应尽可能选择与 CPU 时序相匹配的芯片。

（4）存储器的地址分配和片选译码

微型计算机中的存储器系统通常由 SRAM 类型的 Cache、存储永久信息的只读存储器、保存大量信息的 DRAM 三部分所组成。按空间范围的使用类型来分，存储器系统中又可分为：操作系统保留区域、系统数据区域、设备设置区域、主存储区域和存储扩展区域等。因此，内存的地址分配是一个较为复杂而又必须弄清的问题。另外，由于生产出的存储器的单片容量一般来说总是小于微处理器（或总线控制器）所能寻址的地址范围，所以总是要由许多单片的存储芯片才能组成一个整体的存储系统，这就需要弄清诸多存储芯片之间如何连接，也就是关于如何分配芯片地址、如何产生片选信号的问题。

（5）行选信号 \overline{RAS} 和列选信号 \overline{CAS} 的产生

为了减少芯片的引脚数，DRAM 存储芯片的地址输入采用分时复用方式，输入的地址分成了两部分。高位地址作为行地址，在 \overline{RAS} 的控制下首先送入芯片；然后是低位地址，在 \overline{CAS} 的控制下通过相同的引脚送入芯片。但 CPU 发出的地址码是通过地址总线同时送到存储器的，因此，为了达到芯片地址引脚分时复用的目的，需要专门的存储器控制单元来控制实现。其与 CPU 的连接

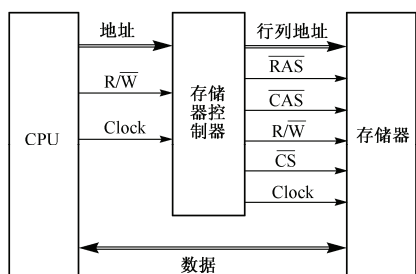


图 4-21 行列地址产生示意图

方式如图 4-21 所示。

存储器控制单元从总线接收完整的地址和控制信号 R/\overline{W} ，将行列地址存储在缓冲器中，并产生 \overline{RAS} 和 \overline{CAS} 信号，因此由该控制单元提供 RAS-CAS 的时序和地址复用功能，也由该控制单元向存储器发出 R/\overline{W} 信号和 \overline{CS} 信号。

对于同步动态存储器芯片，控制单元还要提供时钟信号。一般的动态存储器都没有自动刷新的能力，存储器控制单元还要提供刷新所需的所有信号，如行地址计数等。

4.3.2 主存的逻辑设计

设计存储器时，需先明确所要求的总容量这一技术指标，即总容量=编址单元数×位数。位数指每个编址单元的数据宽度。如果主存按字节编址，那么每个编址单元的宽度为 8 位（1 字节），大多数计算机都允许按字节访问。当计算机字长超过 8 位时，为了提高存取速度，有的主存既允许按字节编址，也允许按字编址。按字编址时则需确定每个字多少位。通常都让字长为字节的整数倍，在 16 位机中就将一个字分为高位字节和低位字节。

然后，需要确定可供选用的存储芯片，即什么型号规格的存储芯片，每片的容量是多少。每片容量通常低于总容量，就需要用若干块芯片组成。在组织芯片的时候，一般原则是：按地址区从低到高，先安排 ROM 芯片后安排 RAM 芯片，先安排大容量芯片其次再安排小容量芯片。相应地，可能存在位数与字数的扩展问题。

① 位扩展。例如，PC/XT 机的主存容量典型值为 $1\text{ M}\times 8\text{ bit}$ ，即 1 MB，典型组成方式是用 8 片 1 Mb（ $1\text{ M}\times 1\text{ 位}$ ）的存储芯片拼接而成。为了实现位扩展，各芯片的数据输入/输出线相拼接绑定，如每片分别与 1 位数据线相连，拼接为 8 位。而编址空间相同的芯片，地址线与片选信号分别相同，可将它们的地址线按位并联然后与地址总线相连，共用一个片选信号。向存储器送出某个地址码，则 8 块存储芯片的某个对应单元同时被选中，可向这 8 块芯片各写入 1 位，或各读出 1 位，拼接为 8 位。

② 编址空间的扩展。如果每片提供的地址数量不够，则需用若干芯片组成总容量较大的存储器，称为编址空间的扩展。为此将高位地址译码产生若干不同片选信号，按各芯片在存储空间分配中所占的编址范围，分送至各芯片。低位地址线直接送至各芯片，以选择片内的某个单元。各芯片的数据线则按位并联到数据总线上。向存储器送出某个地址码，经译码后则只输出一个唯一的有效片选信号，从而只选中某一个芯片，而低位地址在芯片内译码则能选中某一个具体的存储地址单元，该芯片便可写入或读出数据。

在实际的主存储器中。可能只需进行位扩展，也可能即有地址空间扩展又有位扩展。下面通过一个例子介绍存储器逻辑设计的一般方法。

【例 4-3】某半导体存储器容量 $4\text{ K}\times 8\text{ b}$ 。其中固化区容量为 2 KB，拟选用 EPROM 芯片 2716（ $2\text{ K}\times 8\text{ b}$ ）；工作区 2 KB，选用 RAM 芯片 2114（ $1\text{ K}\times 4\text{ b}$ ）。地址总线 $A_{15}\sim A_0$ （低），双向数据总线 $D_7\sim D_0$ （低），读/写控制信号线 R/\overline{W} 。

（1）存储空间分配与芯片

先确定芯片数量，在分配存储空间，以作为片选逻辑的依据。本例即有字扩展也有位扩展，共需 1 块 2716、4 块 2114，每 2 块 2114 拼接为同地址的一组，如表 4-2 所示。

表 4-2 芯片组织及地址分配

| 组号 | 芯片组合 | | $A_{15}A_{14} \dots A_{12} A_{11} A_{10} A_9 A_8 \dots A_1 A_0$ | 地址线 |
|----|-------|-------|--|------------------|
| 0 | 2 K×8 | | $\begin{array}{cccccccc} 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 & 1 & 1 & \dots & 1 & 1 \end{array}$ | 芯片组分配的 起止地址范围 |
| 1 | 1 K×4 | 1 K×4 | $\begin{array}{cccccccc} 0 & 0 & \dots & 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & 1 & 1 & \dots & 1 & 1 \end{array}$ | |
| 2 | 1 K×4 | 1 K×4 | $\begin{array}{cccccccc} 0 & 0 & \dots & 0 & 1 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 1 & 1 & 1 & 1 & \dots & 1 & 1 \end{array}$ | |

(2) 地址分配与片选逻辑

| 组号 | 芯片容量 | 片内地址 | 片选信号 | 片选逻辑 |
|----|------|-------------------|-------------------|----------------------------|
| 0 | 2 K | $A_{10} \sim A_0$ | \overline{CS}_0 | $\overline{A_{11}}$ |
| 1 | 1 K | $A_9 \sim A_0$ | \overline{CS}_1 | $A_{11} \overline{A_{10}}$ |
| 2 | 1 K | $A_9 \sim A_0$ | \overline{CS}_2 | $A_{11} A_{10}$ |

存储器的地址空间是 4 K，共需 12 根地址线 $A_{11} \sim A_0$ 。高 4 线 $A_{15} \sim A_{12}$ 恒为 0，在设计片选信号时可以舍去不用。对于 2716，每片 2 K 个单元，故应将低 11 位地址线 $A_{10} \sim A_0$ 连接到芯片，余下的高位线 A_{11} 作为片选依据。对于两组 2114，每组（由两块拼接）单元数为 1 K，故应将低 10 位地址线 $A_9 \sim A_0$ 连接芯片，余下的两根线 $A_{11} A_{10}$ 作为片选依据。然后再根据存储空间的分配方案，进一步确定各组芯片的片选逻辑。

(3) 存储器设计方案的逻辑结构图

设计的半导体存储器，其连接逻辑如图 4-22 所示。读写命令 R/\overline{W} 送往每个 RAM 芯片，为高电平（ $R/\overline{W}=1$ ）时芯片读出，为低电平（ $R/\overline{W}=0$ ）时写入芯片。2716 的每个存储单元输出 8 位，送往数据总线。每组 2114 中的一片输入/输出高 4 位，另一片输入/输出低 4 位，拼接为 8 位，与数据总线相连。产生片选信号的译码电路，其逻辑关系应满足设计所确定的片选逻辑，注意片选信号一般是低电平有效。

如果存储器中所有存储芯片的容量相同，则设计结果将很规整：加到各存储芯片的地址线相同，产生片选的高位地址位数也相同。此时也可使用通用的译码器芯片，如 2-4 译码器或者 3-8 译码器之类。

如果存储器容量较大，所用的存储芯片采用了地址复用技术，如 2164 一类的芯片，则时序控制逻辑将复杂一些。需按照图 4-14 要求产生一组时序信号，先将高位地址线输入到芯片地址输入端，作为行地址；再将低位地址线输入到芯片地址输入端，作为列地址。在产生片选信号的译码器中，引入行选时序信号 RAS 和列选时序信号 CAS ，译码后产生 \overline{RAS}_0 、 \overline{CAS}_0 、 \overline{RAS}_1 、 \overline{CAS}_1 。

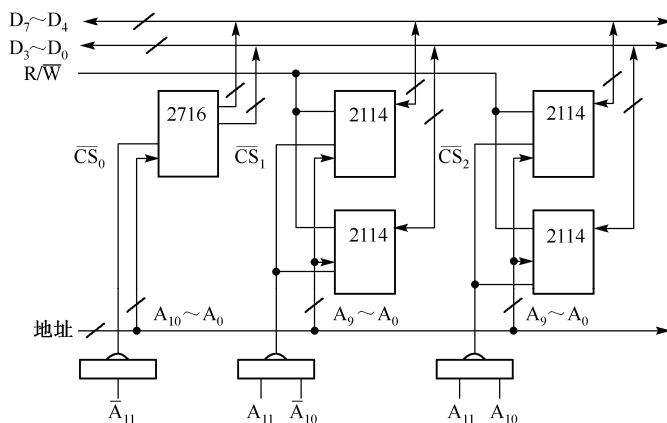


图 4-22 存储器逻辑图

【例 4-4】 设计容量为 14 KB 的半导体存储器，按字节编址，要求 0000H~1FFFH 为 ROM 区，2000H~37FFH 为 RAM 区，地址总线 $A_{15} \sim A_0$ （低），双向数据总线 $D_7 \sim D_0$ （低），读写控制信号线 R/\overline{W} 。可选用的存储芯片规格有 EPROM（4KB/片）和 RAM（2K×4 位/片）。

（1）计算芯片数量

ROM 区容量：(1FFFH-0000H+1) ÷ 2¹⁰=8K，故 EPROM 要 2 片。

RAM 区容量：(37FFH-2000H+1) ÷ 2¹⁰=6K，故 RAM 要 6 片。

（2）确定地址线和片选信号线

存储器按字节编址，即每个地址对应 1 字节，因此 EPROM 芯片不需进行位扩展，但 RAM 芯片宽度只有 4 位，故要先对其进行位扩展，每两片并接成一组，使数据位宽达到 1 字节，共有 3 组 RAM 芯片。扩展以后的芯片及其分组情况，如表 4-3 所示。

表 4-3 芯片组织及地址分配

| 组号 | 芯片组合 | | $A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8A_7 \dots A_0$ | |
|----|------------|----------|---|------------------|
| 0 | EPROM 4K×8 | | 0 0 0 0 0 0 0 0 0 ... 0 | 芯片组分配的 起止地址范围 |
| | | | 0 0 0 0 0 1 1 1 1 ... 1 | |
| 1 | EPROM 4K×8 | | 0 0 0 1 0 0 0 0 0 ... 0 | |
| | | | 0 0 0 1 1 1 1 1 1 ... 1 | |
| 2 | RAM 2K×4 | RAM 2K×4 | 0 0 1 0 0 0 0 0 0 ... 0 | |
| | | | 0 0 1 0 0 1 1 1 1 ... 1 | |
| 3 | RAM 2K×4 | RAM 2K×4 | 0 0 1 0 1 0 0 0 0 ... 0 | |
| | | | 0 0 1 0 1 1 1 1 1 ... 1 | |
| 4 | RAM 2K×4 | RAM 2K×4 | 0 0 1 1 0 0 0 0 0 ... 0 | |
| | | | 0 0 1 1 0 1 1 1 1 ... 1 | |

因例题中明确显示了地址码是 16 位，因此给出的 16 根地址线均要使用。根据每组芯片的容量和地址空间分布情况，可以确定哪些地址线应该用于片内寻址。第 0 组和第 1 组芯片地址空间均为 4K (2¹²=4K)，故其片内寻址需要 12 根地址线。其余各组芯片的地址空间均为 2K (2¹¹=2K)，故它们用于片内寻址的地址线应需要 11 根。除了用于片内寻址的地址线外，剩余的地址线将全部用于片选信号，如下：

| 组号 | 芯片容量 | 片内地址 | 片选信号 | 片选逻辑 |
|----|------|-------------------|-------------------|--|
| 0 | 4K | $A_{11} \sim A_0$ | \overline{CS}_0 | $\overline{A}_{15} \overline{A}_{14} \overline{A}_{13} \overline{A}_{12}$ |
| 1 | 4K | $A_{11} \sim A_0$ | \overline{CS}_1 | $\overline{A}_{15} \overline{A}_{14} \overline{A}_{13} A_{12}$ |
| 2 | 2K | $A_{10} \sim A_0$ | \overline{CS}_2 | $\overline{A}_{15} \overline{A}_{14} A_{13} \overline{A}_{12} \overline{A}_{11}$ |
| 3 | 2K | $A_{10} \sim A_0$ | \overline{CS}_3 | $\overline{A}_{15} \overline{A}_{14} A_{13} \overline{A}_{12} A_{11}$ |
| 4 | 2K | $A_{10} \sim A_0$ | \overline{CS}_4 | $\overline{A}_{15} \overline{A}_{14} A_{13} A_{12} \overline{A}_{11}$ |

（3）存储器设计方案的结构图

如图 4-23，其中四根地址线 $A_{15} \sim A_{12}$ 经译码后，分别形成两个 EPROM 芯片的片选信号输入，5 根地址线 $A_{15} \sim A_{11}$ 经译码后，分别输出形成三组 RAM 芯片的片选信号。

4.3.3 主存的外部连接方式

主存储器的外部连接方式，主要考虑与 CPU 及总线的连接。在具体逻辑形态上，连接方式可能会有多种变化，但从原理上来讲大致需考虑以下几方面。

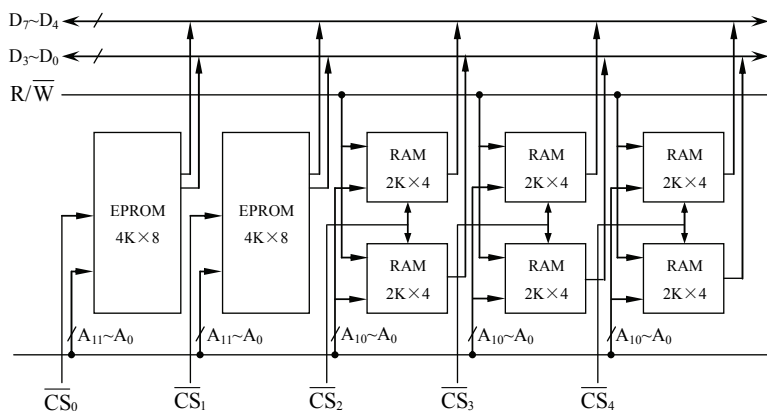


图 4-23 存储器设计方案逻辑结构

(1) 系统的结构模式

① 最小系统模式。将微处理器与半导体存储器做在一块插件上的 CPU 卡，可以作为模块组合式系统中的核心部件，或是多机系统中的一个节点。又如，智能型（可编程控制）设备控制器或接口包含微处理器与半导体存储器。在这些情况下，CPU 芯片与存储芯片直接相连，如图 4-24(a) 所示。CPU 输出地址线直接送往存储器，数据线也直接与存储芯片相连，CPU 还发出读写命令 R/\overline{W} ，送往芯片，称为最小系统模式。由于这种小系统所需存储容量不大，往往采用 SRAM 芯片，省去了刷新逻辑。

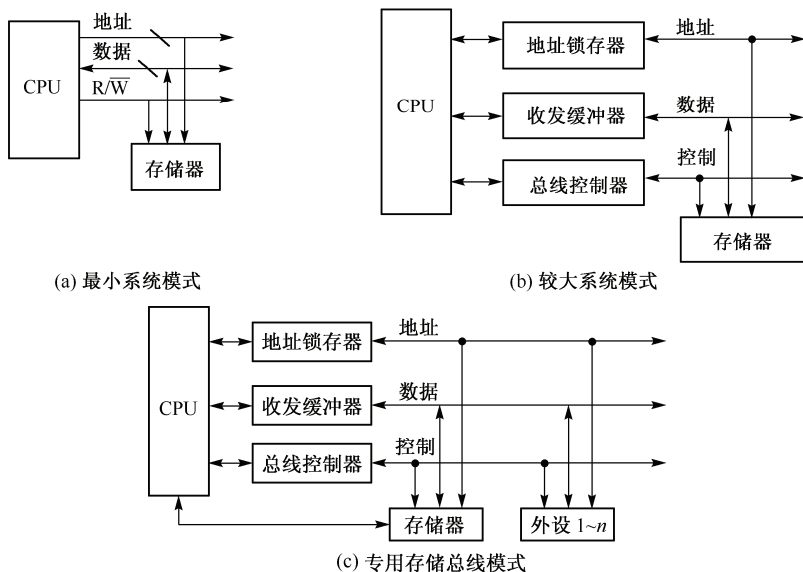


图 4-24 CPU 与主存的连接模式

② 较大系统模式。稍具规模及其以上的计算机系统，都设置了一组甚至多组系统总线，用来连接外围设备。系统总线中包含地址线、数据线及一组控制信号线。CPU 通过数据收发缓冲器、地址锁存器、总线控制器等接口芯片，形成了系统总线。如果主存储器容量较大，需做成专门的存储器模块；或者因速率匹配及其他控制问题，需要配置较复杂的控制逻辑，因而形成独立的存储器模块。可以将主存储器模块挂接于系统总线之上，如图 4-24(b) 所示。完成一次总线传送操作所需的时间，称为一个总线周期。

③ 专用存储总线模式。如果系统规模较大（所带外围设备多）而且要求访存速度较高，可在 CPU 与主存之间建立一组专门的高速存储总线。CPU 既可以通过这组专用总线访问内存，也可以像通过系统总线访问外围设备那样去访问存储器，如图 4-24(c)所示。

（2）速度匹配与时序控制

在早期的计算机中，常为 CPU 内部操作和访存操作设置统一的时钟周期，称为节拍，即以一次访存所需时间为一拍的宽度。CPU 内部操作也是每节拍执行一步。由于 CPU 的执行速度往往高于主存，对 CPU 内部操作来说，时间利用率较低。

现在，大多数计算机对这两类操作设置不同的时间周期。CPU 内将操作时间划分为时钟周期，每个时钟周期完成一步 CPU 内部操作，如一次传输或一次相加。可让时钟频率提高，以适应 CPU 的高速操作。而通过系统总线的一次访存操作，则占用一个总线周期。在同步方式中，一个总线周期可由数个时钟周期组成。大多数主存的存取周期是固定的，因此一个总线周期包含的时钟周期数可以事先确定不变。特殊情况下，也可以安排基本时钟周期数，如果来不及完成读/写，则插入等待（延长）周期。有的系统采用异步方式访存。根据实际需要来确定总线周期的长短，当存储器完成操作时发出一个就绪信号 **READY**，总线周期需长则长，能短则短，与 CPU 时钟周期没有直接关系。高速系统中还采取一种覆盖并行地址传输技术，即在现行总线周期结束之前，提前送出下一总线周期的地址和操作命令。

（3）数据通路匹配

数据总线一次能并行传输的位数，称为总线的通路宽度，常见的有 8 位、16 位、32 位、64 位等。大多数主存储器常采取按字节编址，每次访存读/写 8 位，以应对对字符类信息的处理。这就存在一个主存与数据总线之间的宽度匹配问题，我们通过两个例子说明可能的匹配方法。

【例 4-5】 8088 芯片是一种准 16 位 CPU 芯片，在 CPU 内部可一次处理 16 位（按字），也可一次只处理 8 位（按字节）。对外的数据通路宽度只有 8 位，针对 8088 系统的 PC 总线，其数据总线也只有 8 位。因此，它与主存间的匹配关系比较简单，每个总线周期读写 1 字节，典型时序安排占用 4 个 CPU 时钟周期（ $T_1 \sim T_4$ ），构成一个总线周期。

【例 4-6】 8086 芯片是一种 16 位的 CPU，内部与外部的数据通路宽度都是 16 位。它的标准方式是在一个总线周期可存取 2 字节，即先送出偶单元地址（地址编码为偶数），然后同时读写偶单元与随后的奇单元，用低 8 位数据总线传输偶单元的数据，用高 8 位数据总线传输奇单元的数据。这样的字被称为规则字。如果传输的是非规则字，即从奇单元开始的字，就需要安排两个总线周期。

为了实现例 4-6 的传输，需将存储器分为两个存储体。一个存储体的地址编码均为偶数，称为偶地址（低字节）存储体，与 CPU 低 8 位数据总线相连。另一个存储体的地址编码均为奇数，称为奇地址（高字节）存储体，与高 8 位数据总线相连。如图 4-25 所示，地址线 $A_{19} \sim A_1$ 同时送往两个存储体。每个存储体均有一个选择信号输入端 \overline{SEL} ，低电平选中。体现地址码奇偶的最低位地址 A_0 送往偶地址存储体， A_0 为 0 时选中该体。CPU 输出一个信号 \overline{BHE} （高字节使能），选择奇地址体。

当存取规则字时，地址线送出偶地址、同时让 \overline{BHE} 有效。于是同时选中两个存储体，分别读出高、低字节，共 16 位，在一个总线周期中同时传送。

这种匹配方式可以推广到数据通路宽度更高的系统中，如同时存/取 4 字节，数据总线一次传送 32 位。在 CPU 中，则可以按字处理，也可以按字节处理。

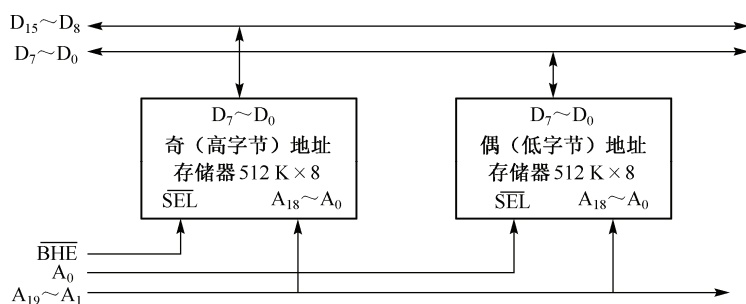


图 4-25 8086 的存储器配置方式

(4) 有关主存的控制信号

如前所述, 存储芯片本身只需要最基本的控制命令, 如 R/\overline{W} 、 \overline{CS} , 或者为实现地址的分时输入将片选分解为 \overline{RAS} 和 \overline{CAS} 。为了实现对存储器的选择、容量扩展、速度匹配, 系统总线可能引申出一些控制和应答信号。不同的系统总线有其自身的约定标准, 规定了一些与主存相关的控制信号, 从而在某种程度上影响了主存储器的整体组织与访存工作方式。下面将提及一些可能遇到的控制信号设置情况。

有的系统总线设置了选择命令 \overline{M}/IO , 低电平时选中主存, 高电平时选中外围设备。相应地, 可将该控制信号引入主存。有的计算机将这一选择信号称为 \overline{MREQ} , 典型的做法是将该信号引至片选译码器的使能端。当 \overline{MREQ} 为高时, 片选译码器的输出无效 (即所有片选均无效, 没有一个存储芯片被选中), 存储器不工作。当 \overline{MREQ} 为低时, 片选译码器有一个片选输出有效, 存储器工作。图 4-25 中的选择信号 \overline{SEL} 作用与此相同。

有的系统总线将存储器选中信号与读写命令结合起来, 分为两个控制信号: \overline{MEMW} (存储器写)、 \overline{MEMR} (存储器读), 将参与控制片选信号的产生, 并形成存储芯片所需的 R/\overline{W} , 或者 \overline{WE} (写) 和 RD (读)。

为了扩展存储器容量, 有的系统允许设置一个基本存储器模板和一个扩展存储器模板, 称为存储器重叠。相应地, 系统总线送出存储器扩展信号 $MEMEX$: 为低电平时, 选择基本存储器模板; 为高电平时, 选择扩展存储器模板。

16 位系统中常设置字节控制信号 \overline{BHE} , 为高电平时选中高字节, 见图 4-25。

主存储器的存取周期一般是已知而且固定的, 因此可用固定的时序信号完成读写, 不需要应答信号。某些特殊情况下, 如主存与外围设备之间的直接传输, 其操作完成时间有可能不固定, 所以需要设置应答信号, 如就绪信号 $READY$ 或传送应答信号 $XACK$ 等。

限于篇幅, 本书不介绍具体的连接逻辑电路, 仅指出几种连接模式。在最小系统中, 主存直接与 CPU 相连, 仅需简单控制信号, 见图 4-24。在较大系统模式中, CPU 将读/写控制命令送往总线控制器 (如 8288 一类芯片), 再由总线控制器送出系统总线信号, 连接到主存中。在有的系统中使用 8203 一类存储器控制器芯片, 总线控制器 8288 产生系统总线控制信号, 送往 8203, 8203 再输出控制信号到存储器。

4.3.4 常见的主存芯片技术

在实际的计算机系统, 特别是在微机系统中, 主存储器通常由多个内存条组合而成。内存条上除记忆信息的存储芯片外, 还存在一些外围电路。在存储器的单个芯片技术中, 除了不断地采用有关先进技术使存储容量不断扩大、存取速度不断提高外, 人们发现在芯片的存储单元之外还

有一些附加的逻辑电路，通过改善和增加少量的额外逻辑电路，可以提高在单位时间内的数据流量，即增加带宽，从而提高芯片的读/写速度。由于所采用的存储原理及半导体工艺、技术的不同，通常能构成多种型号规格的内存条，如 SB SRAM、EDO DRAM、SDRAM、DDR SDRAM 等。

(1) 同步突发静态随机存储器 SB SRAM

一般的 SRAM 是非同步的，为了适应 CPU 越来越快的速度，需要使它的工作时钟脉冲变得与系统同步，这就是 SB (Synchronous Burst, 同步突发) SRAM 产生的原因。SB SRAM 如果作为高性能处理器 (如 Pentium 系列) 的第二级高速缓存，对这种存储器的操作均在统一时钟的控制下同步进行，可配合高性能处理器进行高速的访问操作。

SB SRAM 的主要特点如下：

- ⊙ 支持统一时钟下的同步操作，采用可多次突发访问的多级流水线结构。
- ⊙ 具有片内地址计数器、片内地址缓冲器控制寄存器。
- ⊙ 自定时的写周期。
- ⊙ 既支持按字节写入，也支持全总线宽度写入。
- ⊙ 支持交替突发和线性突发。
- ⊙ 异步输出使能控制。

SB SRAM 主要用于支持按突发地址访问的微处理器系统，所有输入均在时钟信号的上升沿采样。片选信号和输出信号共同控制突发访问的启动和持续。

(2) 多端口静态随机存储器 MP SRAM

MP (Multi-Port, 多端口) SRAM 主要为需要进行数据共享的场合所设计。进行数据共享的多个不同设备需要异步访问保存在同一存储体的信息，因此必须采用这种多端口的存储器。这种存储器一般具有两个独立的端口 (甚至更多端口)。不同端口分别连接不同的设备，通常都具有输出允许 OE#、写允许 WE# 和端口允许 CE# 三个控制信号。

MP SRAM 的存储单元进行了特殊的设计，所以允许从两个端口同时访问某一存储单元。当对两个端口同时进行读操作时，这种设计结构并不要求进行仲裁。但是当对两个端口同时进行读和写操作或同时进行写操作时，则需要进行仲裁。当正在进行读周期的过程中另一个端口发生写周期，读周期就可能读到旧数据或新数据，从而导致读周期所读取数据的不可预见。通过两个端口同时对某一单元进行写操作时，也可能导致存储单元中的数据不可预见。避免产生读写冲突最简单的方法就是执行一次额外的读周期，对于读/写操作所需要进行的仲裁就是使写操作的多组指定地址只通过一个端口输入，以避免冲突的发生。对于指定的多组数据，则通过检查校验和字节来确保正确的数据传输，通过使用一种“邮箱”的软件仲裁系统来传递状态信息。当一个端口进行读操作时，能够为另一个端口的写状态信息指定一个确定的字节，该状态信息能够通知进行读操作的端口以及任何其他端口关于需要激活的信息。

(3) 先进先出存储器 FIFO SRAM

FIFO (First In First Out) SRAM 是一种允许以不同速率进行读/写操作的存储器，其存储体由静态存储器组成，主要作为两种或多种速度不匹配接口电路的中间缓冲。例如，接口电路 A 的传输速率为 n ，接口电路 B 的传输速率为 $n/2$ ，要连接这两种类型的接口电路协调工作，中间就必须加入先进先出存储器 FIFO。假定在 Δt 时间内，接口电路 A 送到接口电路 B 的数据量为 M ，而接口电路 B 在 Δt 时间内送出的数据量只能是 $M/2$ 。另外， $M/2$ 的数据量是来不及送出的，为了保持这剩余的 $M/2$ 个数据不丢失，就必须有保存这些数据的中介存储体，否则这 $M/2$ 个数据就会丢失。这种中介的存储体就由先进先出存储器 FIFO SRAM 来充当。当然，上述接口电路 A 和 B 在

一定的时间内的数据传输总量是一致的,即在 B 完成后续 $M/2$ 个数据量的送出之前,接口电路 A 不能再向接口电路 B 传输数据,直到接口电路 B 将剩余的 $M/2$ 个数据送出。并且,接口电路 A 与接口电路 B 之间的中介存储体 FIFO 的存储容量至少应为 $M/2$ 。

FIFO 存储器广泛用于需要进行速度匹配的场所,如不同总线标准之间的接口电路必须有 FIFO 存储器作为缓冲器。在现代微型计算机中,完成 CPU 主总线到 PCI 总线之间的转换,完成 PCI 到 ISA 总线之间的转换,都采用了 FIFO 存储器。

(4) 扩展数据输出动态随机存储器 EDODRAM

EDO (Extended Data Out) DRAM 与传统的快速面页式动态随机访问存储器 FPM (Fast Page Mode) DRAM 并没有本质上的区别,其内部结构和各种功能操作也与 FPM DRAM 基本相同。主要的区别是:当选择随机的列地址时,如果保持相同的行地址,那么用于行地址的建立和保持时间以及行列地址的复合时间就可不再需要,能够被访问的最大列数则取决于 t_{RAS} 的最长时间。其操作为:先触发内存中的一行,然后触发所需的那一列。但是当找到所需的那条信息时,EDO DRAM 不是将该列变为非触发状态并关闭输出缓冲区,而是将输出数据缓冲区保持开放直到下一列存取或下一读周期开始。因此,EDO DRAM 的存取速度一般比 FPM DRAM 快。

(5) 同步 (Synchronous) 动态随机存储器 SDRAM

SDRAM 是动态存储器系列中新一代的高速、大容量存储器,其内部存储体的单元存储电路仍然是标准的 DRAM 存储体结构,只是在工艺上进行了改进,如功耗更低、集成度更高等。与传统的 DRAM 相比,SDRAM 在存储体的组织方式和对外操作上则表现出较大差别,特别是在对外操作上能够与系统时钟同步操作。

处理器访问 SDRAM 时,SDRAM 的所有输入/输出信号均在系统时钟 CLK 的上升沿被存储器内部电路锁定或输出,即 SDRAM 的地址信号、数据信号及控制信号都是 CLK 的上升沿采样或驱动的。这样做的目的是使 SDRAM 的操作在系统时钟 CLK 的控制下,与系统的高速操作严格同步进行,从而避免因读/写存储器产生“盲目”等待状态,以此来提高存储器的访问速率。

在传统的 DRAM 中,处理器向存储器输出地址和控制信号,说明 DRAM 中某一指定位置的数据应该读出或应该将数据写入某一指定位置,经过一段访问延时后,才可以进行数据的读取或写入。在这段访问延时期间,DRAM 进行内部各种动作,如行列选择、地址译码、数据读出或写入、数据放大等。外部引发访问操作的主控器必须简单地等待这段延时,因此降低了系统的性能。

在访问 SDRAM 时,存储器的各项动作均在系统时钟的控制下完成,处理器或其他主控器执行指令通过地址总线向 SDRAM 输出地址编码信息,SDRAM 中的地址锁存器锁存地址,经过几个时钟周期后,SDRAM 便进行响应。在 SDRAM 响应期间,因对存储器操作的时序确定,处理器或其他主控器能够安全地处理其他任务,而不需简单地等待,因此提高了计算机系统的性能,而且简化了用 SDRAM 进行存储器的应用设计。

在 SDRAM 内部控制逻辑中,采用了一种突发模式,以减小地址的建立时间和第一次访问之后行列预充电时间。在突发模式下,在第一个数据项被访问之后,一系列的数据项能够迅速按时钟同步读出。当进行访问操作时,如果所有要访问的数据项是按顺序访问,并且它们都处于第一次访问之后的相同行中,则这种突发模式非常有效。

另外,SDRAM 内部存储体都采用能够并行操作的分组结构,各分组可以交替地与存储器外部数据总线交换信息,从而提高了整个存储器芯片的访问速率。SDRAM 中还包含特有的模式寄存器和控制逻辑,以配合 SDRAM 适应特殊系统的要求。在 20 世纪 90 年代,台式计算机的内存普遍就是用 SDRAM 存储器芯片构成的,简称 SDR 内存。

(6) 双数据率 (Double Data Rate) 同步动态随机存储器 DDR SDRAM

DDR SDRAM 是在 SDRAM 内存基础上发展而来的, 仍然沿用 SDRAM 生产体系, 因此对于内存厂商而言, 只需对制造普通 SDRAM 的设备稍加改进, 即可实现 DDR 内存的生产, 可有效降低成本。SDRAM 在一个时钟周期内只传输一次数据, 是在时钟的上升期进行数据传输; DDR SDRAM 则在一个时钟周期内传输两次数据, 能够在时钟的上升期和下降期各传输一次数据, 因此称为双倍数据率同步动态随机存储器。DDR SDRAM 可以在与 SDRAM 相同的总线频率下达到更高的数据传输率。

与 SDRAM 相比, DDR SDRAM 运用了更先进的同步电路, 使指定地址、数据的输送和输出等主要步骤既独立执行, 又保持与 CPU 完全同步; DDR SDRAM 使用了 DLL (Delay Locked Loop, 延时锁定回路, 提供一个数据滤波信号) 技术, 当数据有效时, 存储控制器可使用这个数据滤波信号来精确定位数据, 每 16 次输出一, 并重新同步来自不同存储器模块的数据。DDR SDRAM 本质上不需要提高时钟频率就能加倍提高 SDRAM 的速率, 它允许在时钟脉冲的上升沿和下降沿读出数据, 因而其速率是标准 SDRAM 的 2 倍。

从外形体积上, DDR SDRAM 与 SDRAM 相比差别并不大, 它们具有同样的尺寸和同样的引脚距离。但 DDR 为 184 引脚, 比 SDRAM 多出了 16 个引脚, 主要包含了新的控制、时钟、电源和接地等信号。DDR SDRAM 采用的是支持 2.5 V 电压的 SSTL2 标准, 而不是 SDRAM 使用的 3.3 V 电压的 LVTTTL 标准。DDR SDRAM 的频率可以用工作频率和等效频率两种方式表示, 工作频率是内存颗粒实际的工作频率, 但是由于 DDR SDRAM 可以在脉冲的上升和下降沿都传输数据, 因此传输数据的等效频率是工作频率的 2 倍。

继 SDR 内存之后, DDR 内存迅速发展, 2006 年后第一代 DDR 内存逐渐又被 DDR2 内存产品替代。DDR2 是 DDR SDRAM 的第二代产品, 它在 DDR 内存技术的基础上加以改进。与第一代 DDR 内存技术标准最大的不同就是, 虽然都采用了在时钟的上升/下降沿同时进行数据传输的基本方式, 但 DDR2 内存拥有 2 倍于第一代 DDR 内存预读取能力 (即 4 bit 的数据预读取), 换句话说, DDR2 内存每个时钟能够以 4 倍外部总线的速度读/写数据, 并且能够以内部控制总线 4 倍的速度运行, 从而其传输速度更快 (可达 800 MHz), 耗电量更低, 散热性能也更优良。

在 DDR2 内存之后, 逐渐发展出 DDR3 内存。相比 DDR2, DDR3 采用了 8 bit 数据预取设计, 这样 DRAM 内核的频率只有等效接口频率的 1/8, 如 DDR3-800 的核心工作频率 (内核频率) 仅为 100 MHz。DDR3 内存存在达到高带宽的同时, 其功耗反而可以降低, 其核心工作电压从 DDR2 的 1.8 V 降至 1.5 V, 相关数据预测 DDR3 将比现时 DDR2 节省 30% 的功耗。就带宽和功耗之间作个平衡, 对比现有的 DDR2-800 产品, DDR3-800、1066 及 1333 的功耗比分别为 0.72X、0.83X 及 0.95X, 不但带宽大幅提升, 功耗也比 DDR2 更低。在 DDR3 不断发展的同时, 更先进的 DDR4 甚至 DDR5 内存技术标准规范也在逐步完善, 未来的 DDR5 将主要应用于更先进的 GDDR5 规格专用显存, 预计它将比上一代显存具有速度更快、功耗更低、带宽更高等优势。

4.3.5 存储器的刷新与校验

1. 动态随机存储器的刷新

DRAM 芯片是依靠电容上的存储电荷来暂存信息的。电容上存储的电荷会逐渐泄漏, 所以需要定期地进行刷新, 即对原存信息为 1 的电容补充电荷。电荷泄漏速度取决于半导体制造工艺, 如果存储单元没有被刷新, 存储的信息就会丢失。目前, 多数 DRAM 芯片需要在 2 ms 以内全部刷新一遍, 即全部刷新一遍所允许的最大时间间隔为 2 ms, 作为最大刷新周期。如果刷新一遍的

间隔超过 2 ms，电容上的电荷会彻底泄露而导致信息丢失。

对整个存储器来说，各存储芯片可以同时刷新。每块 DRAM 芯片则是按行刷新，每次刷新一行，所需时间为一个刷新周期。如果某存储器有若干块 DRAM 芯片，其中容量最大的一种芯片的行数为 128，则在 2 ms 之中至少应安排 128 个刷新周期。

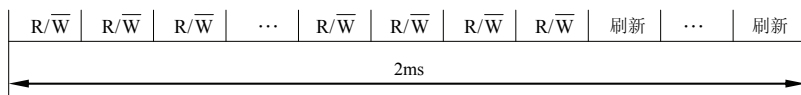
【例 4-7】 某 DRAM 存储芯片，容量为 8KB，组织成 4096 行×16 列，计算应如何安排该芯片的刷新周期 T 。

$T = 2 \text{ ms} \div 4096 \text{ 行} \approx 0.49 \text{ } \mu\text{s} \approx 490 \text{ ns}$ 。即每隔 490 ns 应安排一个刷新周期，完成一行刷新。

前面曾经指出，四管动态存储单元在读出时能自动补充电荷，单管动态存储单元是破坏性读出，但芯片具有读后重写的再生功能。因此只要按行读一次，便实现了对该行的刷新。在刷新周期中，由一个刷新地址计数器提供刷新行的行地址，然后发送行选信号与读命令，此时列选信号 $\overline{\text{CAS}}$ 为高（无效），则可刷新一行，而数据输出呈高阻。每刷新一行后刷新地址计数器加 1，在 2 ms 这一最大间隔内，保证对所有行都至少刷新一次。

于是主存储器需要两种状态。一种是读/写/保持状态，由 CPU（或其他控制器）提供地址进行读写，或者不访问主存；其访存地址根据程序需要是随机的，有些行可能长期不被访问。另一种是刷新状态，由刷新地址计数器逐行地提供行地址，在 2 ms 周期中不能遗漏任何一行。因此，实现动态刷新的一个重要问题是：如何安排刷新周期？这可归纳为下述 3 种典型的刷新方式。

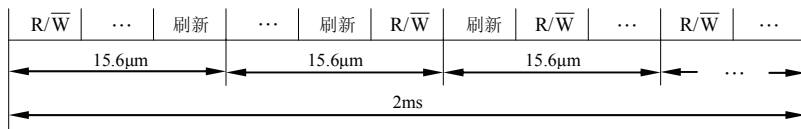
① 集中刷新方式。如图 4-26(a)所示，在 2 ms 间隔内集中地安排若干刷新周期，其余时间可用于正常工作状态，即读、写、保持。刷新周期数等于最大容量芯片的行数。在逻辑实现上，可由一个定时器每 2 ms 请求一次，然后由刷新计数器控制一个计数循环，逐行刷新一遍。



(a) 集中刷新



(b) 分散刷新



(c) 异步刷新

图 4-26 动态刷新的三种方式

集中刷新方式的优点是主存利用率高，控制简单；缺点是在连续、集中的这段刷新周期中不能使用存储器，因而会形成很长一段内存访问的死区。

② 分散刷新方式。如图 4-26(b)所示，将每个存取周期分为两部分，前半期可用于正常读写或保持，后半期用于刷新。也就是将各刷新周期分散地安排在读写周期之后。

分散刷新方式的优点是时序控制简单，主存没有长的死区；缺点是主存利用率不高，速度大约降低一半。这是因为每个存取周期包含一个刷新周期，增加 1 倍时间。现在，微机主存的存取周期约为 100 ns，若采用分散刷新方式，将增至 200 ns，在 2 ms 内将刷新 10^4 次，大大超过行数。因此，分散刷新方式一般只能用于低速系统之中。

③ 异步刷新方式。如图 4-26(c)所示,按行数决定所需刷新周期数,并分散在安排在 2 ms 的刷新间隔中。例如,最大行数为 128,而 $2\text{ ms}/128=15.625\text{ }\mu\text{s}$,则每隔 $15.6\text{ }\mu\text{s}$ 提出一次刷新请求,安排一个刷新周期,完成一行刷新。提出刷新请求时有可能 CPU 正在访存,可稍等待至 CPU 交出控制权后,再安排刷新周期,所以称为异步刷新方式。大多数计算机系统都有一种 DMA 方式(直接存储器访问),可将动态刷新请求作为一种 DMA 请求,由 DMA 控制器控制 DRAM 的刷新,目前这已成为一种典型方式。

异步刷新方式兼有前面两种方式的优点:对主存速率影响最小,甚至可利用不访存的空闲时间进行刷新,因而不会有明显的内存访问死区。虽然控制上复杂一些,但可利用系统的 DMA 功能实现。因此,大多数计算机系统都采用异步刷新方式。

可用通用芯片自行设计动态刷新控制逻辑,也有专用芯片可供利用,如 Intel 8203DRAM 系统控制器包括地址多路转换、地址选通、刷新逻辑、刷新/访存裁决器等,可为 2164、2118、2117 等 DRAM 芯片提供控制信号,有关细节请查阅手册。在有 DMA 功能的计算机系统中,常利用一个 DMA 通道来管理 DRAM 刷新,有关 DMA 方式的工作原理将在后面的章节中详细介绍。

2. 主存储器的校验

从主存中读得的数据是否正确无误对计算机能否正常工作至关重要。因此,需对读出信息进行校验,若发现错误,或给出检验出错的指示信息,或先让主存重读一至数次。如果重读后正确,说明原先的错误是偶然发生的(如受到干扰),现已消失。如果重读后始终有错,说明错误是永久性的,如原存信息已被破坏或主存已产生故障,可能需要停机处理或采取其他故障排除措施。

通俗地讲,校验的方法是让写入的信息符合某种约定的规律,在读出时检验读出信息是否仍符合这一约定规律,如果符合,则基本上可判定读出信息正确无误。

(1) 奇偶校验

大多数主存储器都支持奇偶校验(Parity)方式,这是一种最简单也是应用最广泛的校验方法。奇偶校验是一种总称,其基本思想是根据代码字的奇偶性质进行编码与校验,一般有两种可供选用的编码方式:奇校验(Odd Parity)和偶校验(Even Parity)。带奇偶校验功能的内存一般也被称为奇偶内存,广泛用于普通计算机的主存。关于奇偶校验的更多细节,请参见本书 2.4.3 节。

根据主存是按字节编址或是按字编址的不同,以字节或以字为单位进行编码,每字节(字)配一个奇偶校验位。例如,可用 9 片 1 Mb 的 DRAM 芯片组成 1 MB 主存,增设的 1 位用来作为专用的数据校验位,以保存校验码。

带奇偶校验的主存,存储数据时会有效数据连同校验位一起写入到主存。当读取主存数据时,它会再次按奇偶校验方式计算有效数据的校验位,并判断与之前存储的校验位是否一致。如发现二者不同,就会试图去纠正其可能发生的错误。奇偶校验有个缺点,当检查到主存的数据位有错误时,并不一定能确定具体是哪一位发生了错误,因此就不一定能纠正错误。所以,带有奇偶校验功能的主存一般情况下仅能“发现错误”,理想情况下顶多也只能纠正一部分简单的错误。

计算机中的主存一般以字节(或字)为编址单位,为了确保主存的读写速度,在系统层面基本上都是依靠硬件电路来实现主存奇偶校验的编码和检错。除此之外,有时在应用程序层面还可以通过软件方式来实现“累加和”以进行校验编码和检错。

(2) ECC 校验

ECC(Error Checking and Correcting, 错误检查和纠正)是另外一种继奇偶校验之后发展起来的校验技术,具有更高的编码效率和更强的自动纠错能力。

行业中一般将具备 ECC 功能的主存称为 ECC 主存,表明它具备 ECC 方式的检错和纠错。这

种内存一般多应用在高档台式计算机/服务器及图形工作站上,它能使整个计算机系统在工作时更加安全和稳定。

ECC 内存同样是在有效数据位上额外增加校验位来实现错误检测和纠正的。当数据写入到内存时,有效数据位和对应的 ECC 校验码同时被写入到内存中。读数据时,根据读取的有效数据位重新计算 ECC 校验码,并将其与存储的 ECC 码进行比较。如果两个 ECC 码相同,则表明数据正确,系统正常执行。如果两个 ECC 码不同,则认为数据发生错误,于是通过 ECC 检错规则确定错误位并对其进行纠正。

使用 ECC 方式来校验的内存,会对系统的性能造成一定影响,但这种检错和纠错对确保计算机的可靠性十分必要。此外,由于带 ECC 校验功能的内存要比普通内存的价格昂贵很多,因此 ECC 内存一般都只在高端计算机使用,如高性能笔记本和服务器等。

4.4 磁表面存储原理

磁表面存储器是目前使用最广泛的外存储器。在可预测的一段时间内,它仍将占有重要地位。根据记录载体(介质及基体)的外形,磁表面存储器有磁鼓、磁带、磁盘、磁卡等。由于外形尺寸大而记录面积有限,磁鼓已被淘汰。磁卡的记录信息量较少,主要用于某些专用设备之中。因此,在计算机系统中广泛使用的是磁盘和磁带,特别是磁盘,几乎是稍具规模的系统的基本配置。

本章主要介绍磁表面存储器的基本存储原理,如记录介质和磁头、读写原理、磁记录编码方式,以及在外存储器中采用的校验方法等。

4.4.1 存储介质与磁头

(1) 基体与磁层

在磁表面存储器中,记录信息的介质是一层很薄的磁层,需要依附于具有一定机械强度的基体之上。根据不同磁表面存储器的需要,基体可分为软质基体和硬质基体两大类,它们所要求的磁层材料和制造工艺也相应不同。

① 软质基体与磁层。磁带的运行方式要求采用软质基体,如聚酯薄膜带。软盘的盘片在工作时与磁头接触,为减少磁头磨损,也要求用软质基体,如聚酯薄片。

将具有矩磁特性的氧化铁微粒,渗入少量钴,用树脂黏合剂混合后,涂敷在基体之上加工后形成约 $1\text{ }\mu\text{m}$ (微米)厚的均匀磁层。这就是记录信息用介质,属于颗粒型材料。

② 硬质基体与磁层。硬盘的运行方式对基体和磁层要求更高,一般采用铝合金硬质盘片作基体。为了进一步提高盘片光洁度和硬度,一些新型硬盘采用工程塑料、陶瓷、玻璃作为基体。

硬盘一般采用电镀工艺在盘片上形成一个很薄的磁层,所用材料为具有矩磁特性的铁镍钴合金。电镀形成的磁层属于连续型非颗粒材料,又称为薄膜介质,其均匀性和性能大为提高。磁层厚度大约有 $0.1\sim 0.2\text{ }\mu\text{m}$,上面再镀一层保护层,增加抗磨性和抗腐蚀性。

在更新的硬盘中,采用溅射工艺形成薄膜磁层,即用离子撞击阴极,使阴极处的磁性材料原子淀积为磁性薄膜。其性能优于镀膜。

为了增加读出信号的幅度,希望选用材料的剩磁感应强度 B_Y 比较大。但 B_Y 过大,磁化状态翻转时间增加,因而影响记录密度。为了提高记录密度,要求磁层尽量薄,以减少磁化所需时间。但磁层薄又使磁通变化量 $\Delta\Phi$ 减少,将影响读出信号幅度。这就要求改进读出放大的电子技术,以降低对磁层制造工艺要求,或在相同工艺水平条件下,提高密度和可靠性。

此外, 要求磁层内部无缺陷, 表面组织致密、光滑、平整, 磁层厚薄均匀, 无污染, 对环境温度不敏感, 性能稳定。

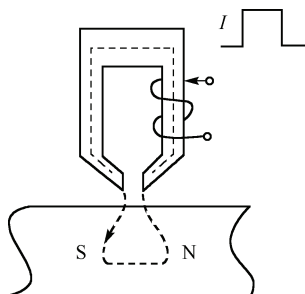


图 4-27 磁头原理示意图

(2) 读写磁头

磁头是实现读写的关键元件。写入时, 将脉冲代码以磁化电流形式加入磁头线圈, 使记录介质产生相应的磁化状态, 即电磁转换。读出时, 磁层中的磁化翻转使磁头的读出线圈产生感应信号, 即磁电转换。

图 4-27 是磁头的原理示意图。磁头由高导磁材料构成, 上面绕有线圈。由一个线圈兼作写入磁化与读出, 或分设读磁头与写磁头。

磁头面向记录介质的部分开有间隙, 称为磁头间隙, 简称头隙。如果没有这个间隙, 磁化电流产生的磁通将只在闭合磁路中流过, 对记录介质没有作用。开了间隙后, 大部分磁通将流经头隙所对应的记录介质局部区域, 使该作用区留下某种磁化状态。读出时, 记录了信息的介质经过磁头, 由于正对磁头的区域中存在磁化状态翻转, 若由正向饱和变为负向饱和, 或由负向饱和变为正向饱和, 使磁头的磁路中发生磁通变化 $\Delta\Phi$ 。读出线圈产生感应电势, 即读出信号。因此头隙部分的形状和尺寸至关重要, 又称为工作间隙, 而磁头的磁路其余部分既可做成圆环形, 也可做成马蹄形, 影响不大。

在磁盘或磁带进行读/写时, 记录介质运动而磁头不动, 磁头在记录介质上的磁化区形成磁道。磁化后, 磁道中心部分可达到磁饱和, 而磁道两侧的边缘部分磁化不足。在写入后, 常将两侧进行清洗, 称为夹缝清除。

从磁头任务来看, 在磁盘中, 每个记录面有一个磁头, 兼作读磁头和写磁头, 又称为复合磁头。在磁带机中, 常一次并行地读写几个磁道。每个磁道中有一对磁头: 一个读磁头和一个写磁头, 可实现写后读出检查。将几个磁道的读磁头和写磁头装配为一体, 道间加屏蔽, 称为组合头块。

从制造工艺方面看, 可分为早期的传统工艺磁头和近期的薄膜磁头。

在早期的制造工艺中, 或用高导磁率铁淦氧材料热压成形, 或用高导磁率铁镍合金(坡莫合金)叠片组装成形。通常是先制成几部分, 其中一段绕有线圈, 再将它们粘接起来。用于软盘的磁头将上述铁芯封装在特种塑料外壳中。外壳做成球面形或平面扣子形, 便于安装和定位, 并使磁头与盘面接触良好, 工作时磨损小。用于硬盘的磁头将铁芯封装在一个陶瓷块中, 该陶瓷块称为浮动块, 工作时可由气垫使其浮空于盘面上; 后来又将铁芯和浮动块改为用同样的材料制成。

近期的硬盘采用薄膜磁头, 用类似于半导体工艺的淀积和成形技术, 在基板上形成坡莫合金的铁芯和具有一定匝数的线圈, 如平面螺旋式导体线圈。制造成形过程中使用掩模光刻技术精度很高, 可获得比较理想的极尖形状和工作间隙。然后在基板上烧固一层氧化铝和碳化钛, 再切割加工成浮动块。相比之下, 薄膜磁头在各方面的性能均优于传统工艺的磁头。

4.4.2 读写原理

为了写入不同的数据, 磁化电流按一定编码方法呈波形变化, 且随时间而变。在写入或读出数据的过程中, 记录介质与磁头之间做相对运动, 一般是记录介质运动而磁头不动。磁表面存储器通过磁层的不同磁化状态来存储信息, 常见的磁化方式主要有两种, 即水平磁化方式和垂直磁化方式, 如图 4-28 所示。

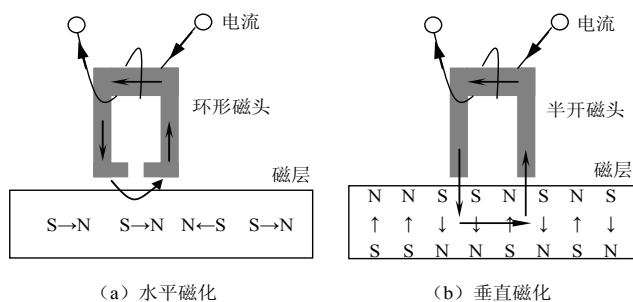


图 4-28 两种不同的磁化方式

1. 水平磁化方式

如图 4-28(b)所示, 水平磁化方式可采取分解不同时刻的电流变化、磁化状态、留下的剩磁状况、读出时的感应电势等来分析其原理, 如图 4-29 所示。

(1) 写入 (位单元的形成)

在 $t=t_0$ 时, 线圈中流过正向电流 $I=+I_m$, 则磁头下方将出现一个与此对应的磁化区。磁通进入磁层的一侧为 S 极, 离开磁层的一侧为 N 极。如果磁化电流足够大, S 极与 N 极之间被磁化到正向磁饱和, 以后将留下剩磁 $+B_r$, 用箭头 \rightarrow 表示。由于磁层是矩磁材料, 剩磁 B_r 的大小与饱和磁感应强度 B_m 相差无几。

$t_0 \rightarrow t'_1$ (电流方向变化前), 由于记录磁层向左运动, 而磁化电流维持 $+I_m$ 不变, 相应地出现图 4-29(b)所示磁化状态。即 S 极左移一段距离 ΔL , N 极仍位于磁头作用区右侧不变。

在 $t=t_1$ 时, 则磁化电流改变方向, $I=-I_m$, 相应地磁层中的磁化状态也出现翻转, 如图 4-28(c)所示。已移离磁头作用区的 S 极以及一段 $+B_r$ 区, 维持原来磁化状态不变 (剩磁)。而磁头作用区下出现新的磁化区, 左侧为 N 极, 右侧为 S 极, N、S 之间是负向磁饱和区 $-B_r$, 用箭头 \leftarrow 表示。

在写入电流的作用下, 于是在记录磁层中留下一个对应于 $t_0 \rightarrow t_1$ 的位单元, 其起始处和结束处两侧各有一个磁化状态的转变区。根据转变区的存在及其性质 (位置、方向、频率等), 体现出所存储的信息。后面再分析信息代码 (0、1) 与转变区之间的关系, 这与所采用的磁记录编码方式有关。

(2) 读出

读出时, 磁头线圈不加磁化电流, 作为读出线圈使用。当已经磁化的记录磁层位于磁头下方时, 由于铁芯部分的磁阻远小于头隙磁阻, 则记录磁层与磁头铁芯形成一个闭合磁路, 大部分磁通将流经铁芯再回到磁层。如果记录磁层在磁头下方运动, 则各位单元将依次经过磁头下方。每当有转变区经过磁头下方时, 铁芯中的磁通方向也随之改变, 在读出线圈产生相应的感应电势。

$$e = -\frac{d\Phi}{dt}$$

感应电势 e 即读出信号, 它的方向取决于记录磁层转变区方向 (由 $-B_r$ 变为 $+B_r$, 或者由 $+B_r$

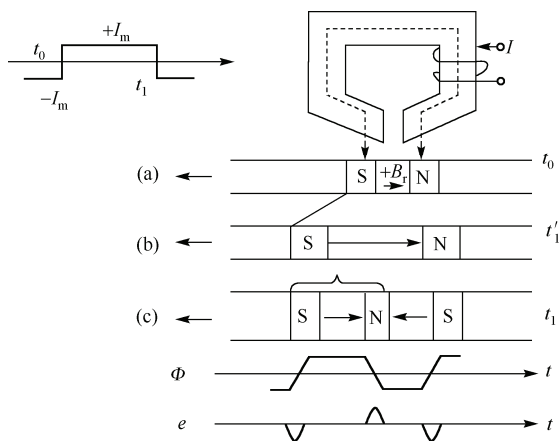


图 4-29 读写过程示意图

变为 $-B_r$), 其幅值大小则与 B_r 值有关 (最大变化量 $2B_r$)。

如果记录磁层中没有转变区, 维持一种剩磁状态 ($-B_r$ 或 $+B_r$), 则磁层经过磁头下方时, 铁芯中磁通量没有变化, 也就没有读出信号。

2. 垂直磁化方式

垂直记录技术是新一代的硬盘存储技术, 如图 4-28(b)所示, 通过把原本横向放置的磁记录单元变为纵向排列, 减少每个磁单元所占用的表面积, 从而提高存储密度增加存储容量。与水平磁化的存储器设计相比, 垂直记录技术的数据存储密度几乎翻了一倍, 每平方英寸的面积上甚至可以存储高达 230 GB 的数据。

垂直磁化技术是丹麦科学家 Valdemar Poulsen 在 19 世纪后期首创, 他被业界公认为第一个能利用垂直记录技术在磁带上记录声音的人。直到 1976 年, 日本东北工业大学岩崎俊一 (Shun-ichi Iwasaki, 垂直记录之父) 教授在这项技术上又实现了重大突破。从那以后, 业界在研究水平磁化技术的同时, 对垂直磁化技术的研究和探索从未停止。2005 年, 希捷公司推出第一块真正意义上的垂直磁化商用硬盘, 此后垂直磁化方式存储数据的硬盘便逐渐成为了主流。

应用了垂直磁化记录数据的硬盘在结构上不会有什么明显变化, 依然是由磁盘 (超平滑表面、薄磁涂层、保护涂层、表面润滑剂)、传导写入元件 (软磁极、铜写入线圈) 和磁阻读出元件 (检测磁变换的 GMR 传感器或磁盘者新型传感器设计) 组成。从磁层的微观细节上看, 垂直磁化方式下磁记录单元的排列方式有根本性的变化, 它已从原来水平磁化 “首尾相接” 式的水平排列变为了 “肩并肩” 式的垂直排列。此外, 磁头的构造也有了改进, 并且增加了软磁底层, 这样做的优点是: ① 磁盘材料的厚度增厚, 使微型磁粒更能抵御 “超顺磁效应” 的不利影响; ② 软磁底层让磁头可以提供更强的磁场, 让其能够以更高的稳定性将数据写入存储介质; ③ 相邻的垂直比特可以长时间保持相互稳定。

根据对上述两种磁化原理的分析, 可归纳出磁表面存储器具有如下特点:

- ◎ 记录信息可以长期保存, 属于非易失性存储器 (原则上允许记录介质脱机保存, 但要注意防止外界强磁场破坏其剩磁状态)。
- ◎ 非破坏性读出, 读出不影响原存储信息。
- ◎ 记录介质可以重复使用。
- ◎ 连续记录数据, 所以基本是顺序存取方式, 不能如 RAM 那样随机访问。
- ◎ 由于是连续记录, 需要比较复杂的寻址定位结构配合。
- ◎ 由于是在相对运动中读写, 可靠性低于半导体存储器, 需要更复杂的校验技术。

4.4.3 磁记录的编码方式

前面只讨论了磁表面存储器中读写过程的电磁转换原理, 究竟什么是 0, 什么是 1? 这取决于采用何种磁记录编码方式。磁记录方式就是采用某种变换规律, 将一串二进制代码序列转换成记录磁层中相应的磁化状态。具体地讲, 是如何按照写入代码序列形成相应的写入 (磁化) 电流波形, 还派生出相应的读出识别方法。

记录方式对提高记录密度至关重要。为此出现了多种记录方式, 至今仍在探索改进之中。它的演变思想对我们颇有启发价值。

人们习惯于用两种彼此相反的静止状态去分别表示 0 或 1, 如用负向磁饱和状态表示 0, 用正向磁饱和状态表示 1。相应地, 在写 0 时让磁化电流为 $-I$, 在写 1 时让磁化电流为 $+I$ 。这就派

生出两种传统的磁记录方式：归零制（RZ）和不归零制（NRZ）。归零制的基本点是：写 0 时，先发 $-I$ 电流脉冲，然后回到零（ $I=0$ ）；写 1 时，先发 $+I$ 脉冲，然后归零。实际上这种归零是不必要的，导致读出信号幅度小、转变区数目过多，所以不利于记录密度的提高。不归零制的基本点是：写 0 时维持 $-I$ 不变，不归零；写 1 时维持 $+I$ 不变，也不归零。换句话说，改变写入内容时才改变磁化电流方向，只有信息变换（由 0 \rightarrow 1，或由 1 \rightarrow 0）时才在磁层中产生转变区。转变区数目少，有利于提高记录密度。但一连串相同代码序列（如一串 0 或一串 1）时，没有转变区，无法分辨有几位连续 0 或几位连续 1，一旦因干扰发生一位错，错误将会延伸。

为了克服传统记录方式的缺点，人们摆脱了传统观念的束缚，改以动态观念去设计磁记录方式。例如，以电流方向的变与不变、相位变化的不同、频率变化的不同，来表示和区分 0 和 1，这就出现了实用的不归零-1 制、调相制、调频制。再进一步，我们不必孤立地用一个位单元对应一位代码，也可将一串位单元整体地对应一串代码序列，从而导致一些新的记录编码方式，如改进型调频制、群码制及一些更新的记录编码方式，并发展了游程长度受限码理论。

下面介绍几种常见的记录方式。

1. 不归零-1 制（No Return to Zero-1, NRZ1）

不归零-1 制是在不归零制的基础上改进形成的，其写入规律可概括为：电流见 1 则翻转。

写 0 时，写入电流维持原方向不变（ $-I$ 或 $+I$ ）。写 1 时，写入电流方向翻转（由 $-I \rightarrow +I$ ，或由 $+I \rightarrow -I$ ）。如图 4-30 所示，欲写入代码 001101，假定起始状态为 $+I$ 。写入头两个 0 时，电流维持 $+I$ 不变（可见是以“不变”为 0，不一定是负向饱和，也可能是正向饱和）。每写入一个 1 时，电流改变一次方向，既可能是由 $+I \rightarrow -I$ ，也可能是 $-I \rightarrow +I$ ，因此是以“变”为 1。相应地，每写一个 1，就在磁层中留下一个转变区。

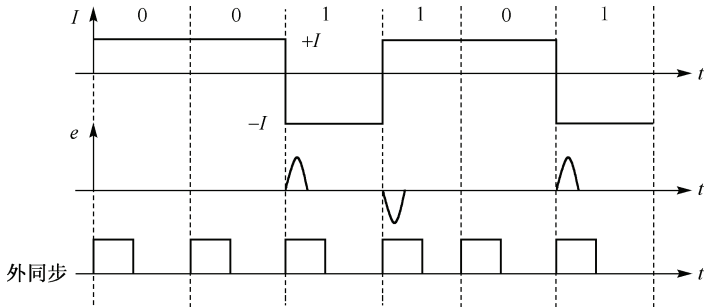


图 4-30 NRZ1 制写入电流波形

读出时，逢 0 没有读出信号，逢 1 有转变区，就有读出的感应电势；即有读出信号为 1，无读出信号为 0。

写入一串代码，NRZ1 制产生的转变区数较少，在同样的技术条件下可以缩短位单元长度，因而可以提高记录密度。这是 NRZ1 制的主要优点。

但在 NRZ1 制中，读一连串 0 时，由于没有转变区存在而没有读出信号，如何识别这是几位 0 呢？这就需要外加同步信号来辨识各位单元，称为外同步方式。换句话说，NRZ1 制没有自同步能力。因此，NRZ1 制不能直接用于像磁盘这种单道记录方式中，但可用于像磁带这种同时读写多道的设备之中。有两种方法可产生外同步信号：一种是在磁带上专门写入一个同步信号道，每位均为 1，即每位均有一个转变区，读出时每位都产生一个同步信号，用以选通数据道的各位；另一种方法是不设专门的同步道，而是让同时读写的各位（称为一个带字）采取奇校验，则每个

带字中至少有一个 1，可以提取出来作为同步信号。

采取外同步方法限制了记录密度的提高，这是 NRZ1 制的主要缺点。因为磁带在运动中难免存在扭斜，各位并不总是准确地同在一根垂直线上（与磁带运转方向垂直）。当记录密度较高时，外同步信号就难以准确地选通其他各位。

为了进一步提高记录密度，要求读出信号序列具有自同步能力，即能从自身读出信号序列中提取同步信号的能力，不需外加同步信号。例如，让每个位单元都至少产生一个转变区，改用转变区变化方向或变化频率的不同区分 0 或 1，这就出现了调相制与调频制。

NRZ1 制曾直接应用在早期低速磁带机中，现在它仍是多种记录方式的基础或中间形式，这一点在后面还会提及。

2. 调相制（Phase Modulation, PM）

如图 4-31 所示，调相制又称为相位编码（Phase Encoding, PE），其写入规律如下：

- ⊙ 写 0，在位单元中间位置让写入电流负跳变，由 $+I \rightarrow -I$ 。
- ⊙ 写 1，在位单元中间位置让写入电流正跳变，由 $-I \rightarrow +I$ 。

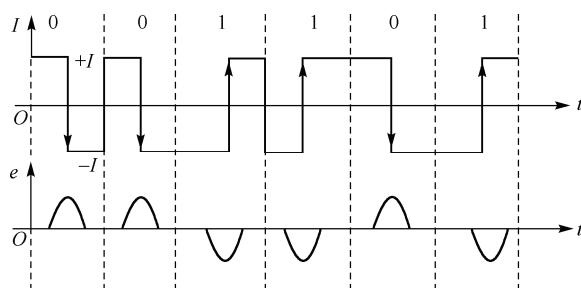


图 4-31 调相制波形

由此派生出一个结论：若相邻两位相同（连续两个 0，或连续两个 1），则两位交界处写入电流需改变一次方向，才能使相同两位的磁化翻转相位一致；若相邻两位不同（01 或 10），交界处没有翻转。

在这样的写入电流作用下，记录磁层中每个位单元中将有一次基本的磁通翻转（0 与 1 的翻转方向不同），即用相位的不同来区分 0 与 1，所以又称为相位编码。

读出时，位单元中间的转变区将产生读出信号。它既是数据信号，也是同步信号，所以调相制具有自同步能力。根据读出信号的相位可识别所存信息在该位是 0 还是 1。如果某位信息丢失，该位没有读出信号，而读出 0 和读出 1 都有读出信号，仅相位相反。可见，在调相制中信息丢失与 0、1 有明显区别，可靠性较高。此外在读出电路中，位单元交界处可能产生的感应电动势被弃之不用。

调相制因具有自同步能力，记录密度可做得比 NRZ1 制高，广泛用于常规磁带机中。

3. 调频制（Frequency Modulation, FM）

如图 4-32 所示，调频制的写入规律如下：

- ⊙ 每个位单元起始处写入电流都改变一次方向，留下一个转变区，作为本位的同步信号。
- ⊙ 在位单元中间记录数据信息。如果写入 0，则位单元中间不变；如果写入 1，则写入电流在位单元中间改变一次方向。

对比写入 0 与写入 1：写 0 时每个位单元只变一次，写 1 时每个位单元变化两次；即用变化

频率的不同来区分 0 与 1，所以称为调频制（FM）。写 1 时频率是写 0 的 2 倍，所以又称为倍频制或双频制（DM）。

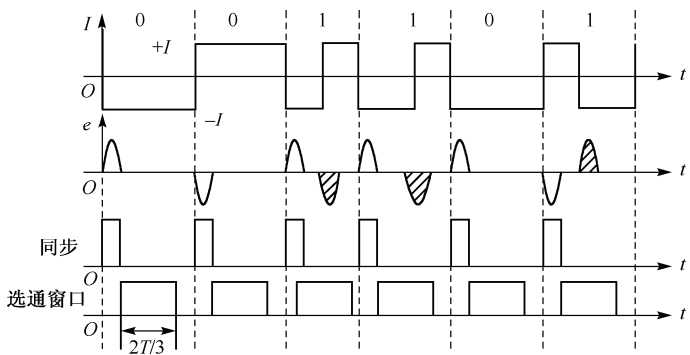


图 4-32 调频制波形

读出时，每个转变区都将产生一个感应电势，所以读出信号序列中包含了同步信号和数据信号。通过分离电路，将每个位单元起始处的信号分离出来，作为该位的同步信号。它将触发一个单稳电路，宽度为 $2T/3$ （ T 为位单元周期宽度），形成一个选通窗口，用于选通位单元中部的读出信号。若在位单元中部有读出，该位为 1；若在位单元中部无读出信号，该位为 0。因此位单元中间的读出信号即数据信号。

显然，调频制能从读出信号序列中提取同步信号，具有自同步能力。记录密度可做得比 NRZ1 制高，广泛用于早期的磁盘机中，是磁盘记录方式的基础。

4. 改进型调频制（Modified FM，MFM 或 M^2F ）

调相制和调频制都属于位间无关型的按位编码。为了获得自同步能力，不一定要每位都有转变区，还可以将记录序列中相邻位联系起来，即采取位间相关型编码，既有一定自同步能力，又进一步减少转变区数，从而提高记录密度。按照这一思路，可对调频制进行改进。

我们将调频制的写入电流波形重画于图 4-32 上半部，分析一下哪些转变区需要保留，哪些转变区可以省去。写 1 时，位单元中间的转变区用来表示数据 1 的存在，因此它应当保留，但位单元交界处的转变区就可以省去。连续两个 0 都没有位单元中间的转变区，所以它们的交界处应当有一个转变区，以产生同步信号。

因此，改进型调频制的写入规律如下：

- ⊙ 写 1 时，在位单元中间改变写入电流方向。
- ⊙ 写入两个以上 0 时，在它们的交界处改变写入电流方向。

如图 4-33 所示，为记录相同代码， M^2F 的转变区数约为 FM 的一半，在相同技术条件下， M^2F 位单元长度可缩短为 FM 的一半，使 M^2F 的记录密度提高近一倍。所以，常将 FM 制称为单密度方式，将 M^2F 制称为双密度方式。 M^2F 制广泛用于软盘和小容量硬盘之中。

5. 群码制（GCR）

群码制即成组编码方式。NRZ1 制的优点是转变区数少，问题在于如何识别一连串 0。如果在编码中作出某种限制，让连续 0 的个数不超过两个，则 NRZ1 制的缺点就可以克服。

在数据流式磁带机中，广泛采用 GCR(4/5) 编码。它的基本方法是将 4 位一组的数据码，整体转换成 5 位一组的记录码；在数据码中连续 0 的个数不受限制，但在转换后的记录码中，连续 0 的个数不超过两个；将转换后的记录码按 NRZ1 制记入磁带中。从信息量的角度看，从 4 位扩大

为 5 位，组合数增加了 1 倍，可以只选取其中连续 0 的个数不超过 2 的组合，将连续 0 的个数在 2 以上的组合丢弃不用。具体变换规律如表 4-4 所示。

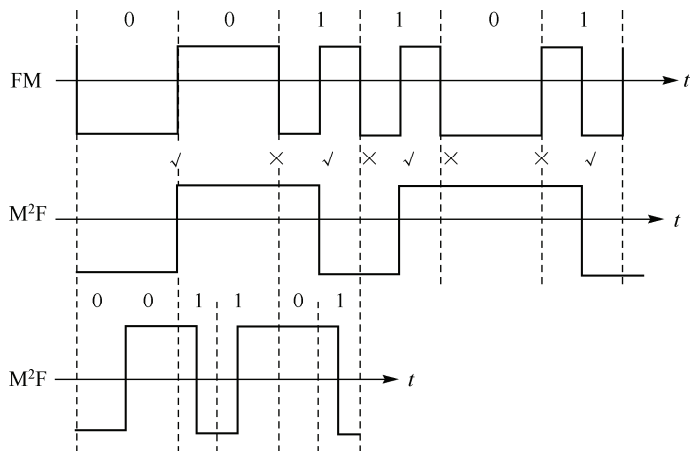


图 4-33 改进型调频制波形

表 4-4 GCR(4/5)转换规律

| 数据码 | 记录码 | 数据码 | 记录码 | 数据码 | 记录码 |
|------|-------|------|-------|------|-------|
| 0000 | 11001 | 0110 | 10110 | 1100 | 11110 |
| 0001 | 11011 | 0111 | 10111 | 1101 | 01101 |
| 0010 | 10010 | 1000 | 11010 | 1110 | 01110 |
| 0011 | 10011 | 1001 | 01001 | 1111 | 01111 |
| 0100 | 11101 | 1010 | 01010 | | |
| 0101 | 10101 | 1011 | 01011 | | |

以上介绍了 5 种常见的磁记录方式，简明地说明了编码规律与基本概念。事实上，在磁记录编码理论研究中提出了一个通用的概念：游程长度受限码 RLLC (Run Length Limited Code)，上述编码方式也可纳入这一概念中进行分析。游程是指在数据序列中一串连续 0 的个数。按 RLLC 编码形成的记录序列，可用 5 个参数描述其结构特性：数据码长度 m 位，记录序列长度 n 位 ($m < n$)，记录序列中两个 1 之间至少存在 d 个 0，最多存在 k 个 0，一次变换的最大数据长度与最小数据长度之比值 r 。游程长度受限即是指对 d 与 k 的值作出限制。

【例 4-8】 NRZI 制不是游程长度受限码。

【例 4-9】 GCR(4/5)属于游程长度受限码。 $m=4$ ， $n=5$ ， $d=0$ ， $k=2$ ， $r=1$ （每次变换的数据长都是 4 位）。

根据 RLLC 理论，又推出了许多新型磁记录方式，如“3PM”、“2,7RLLC”等。

4.5 磁盘存储器及其接口

磁盘存储器作为磁表面存储技术的典型应用产品，不仅是目前应用最广泛的输入/输出设备，也是计算机存储体系结构中绝不可缺少的外部存储器。

典型的磁盘子系统包括下述部分。

(1) 硬件

- ◎ 盘片（存储体）。
- ◎ 磁盘驱动器。在软盘存储器中，盘片可以拆卸，所以盘片可与驱动器相分离。在许多硬盘

存储器中，盘片常被密封地组装于驱动器之中，因此一般情况下不可被分离，硬盘驱动器就包括了存储信息的盘片。

- ◎ 磁盘控制器与接口。磁盘控制器是用来控制磁盘驱动器工作的，而接口是主机与磁盘存储器之间的连接逻辑。在常见的微机系统中，磁盘控制器与接口往往制作在一块插件上，称为磁盘适配卡。

(2) 软件

磁盘驱动程序是操作系统软件中的设备管理程序。在微机系统中，这部分常固化于系统板的 ROM 之中。磁盘控制程序是磁盘控制器中的软件，实现磁盘驱动器的有关操作。在微机系统中，这部分常固化于磁盘适配卡的 ROM 中。

操作系统提供了用户界面，如键盘命令、系统功能调用命令，用户可以按文件名操作。如从某盘中调出某个文件到主存，或从主存将某文件存入某磁盘等。在这个层次上，用户不必过问该文件究竟在磁盘中的哪一物理存储区中。

操作系统中的文件系统，对用户发出的文件操作命令进行解释，通过文件目录检索找到该文件所在的物理位置，然后启动设备管理程序。操作系统中的设备管理程序一般包含：I/O 调度程序、设备读/写管理程序、缓冲区管理程序（在主存中开辟相应的缓冲区）、设备驱动程序。在磁盘驱动程序控制下，向磁盘适配器发出操作命令、回收有关状态信息、接收中断请求与 DMA（直接存储器访问）请求、收发数据信息等。在驱动程序这一级，将涉及磁盘存储器的物理特性与寻址信息，如驱动器的存储容量、记录面数、磁道数，以及驱动器号、记录面号、磁道号、起始扇区（数据块）号、需传输的扇区数等寻址信息。

现在的磁盘控制器常属于所谓智能控制器型，内含微处理器，可以执行控制程序。它向驱动器发出控制命令，如选择驱动器、磁头、寻道方向与寻道信息、读/写命令等；接收驱动器有关状态信息，如选中回答、准备就绪、寻道完成、0 道检测、索引信号等。磁盘控制器将写入数据编成某种记录码序列，发送到驱动器，以产生写入电流。从驱动器获得读出序列，从中分离与提取同步信号与数据信号。

本节主要介绍磁盘存储器的基本结构原理、信息分布方式（重点）、磁道记录格式和主要特性，以软盘和硬盘为主。

4.5.1 软盘存储器

1972 年正式推出的软盘存储器，20 世纪 80~90 年代成为大多数计算机系统的基本配置，但目前，优盘（或称 U 盘）、存储卡和移动硬盘等已全面取代了软盘。与硬盘存储器相比，软盘存储器的结构比较简单、制造成本低、盘片可拆卸，可以脱机保存信息；但记录密度较低，存储容量较小，使用寿命也较硬盘短。

1. 盘片结构与分类

(1) 盘片结构

如图 4-34 所示，软盘盘片是一种圆形盘片，盘片装在纸质保护套中，使用时一起插入软盘驱动器中。工作时保护套不动，盘片旋转。盘片以软质聚酯塑料薄片为基体，在基体上涂敷氧化铁磁性材料作为记录介质，靠近轴心的一个区域及外周边缘是空白区。以 3 英寸软盘片为例，记录区宽度约为 2 cm；保护套上开了一个

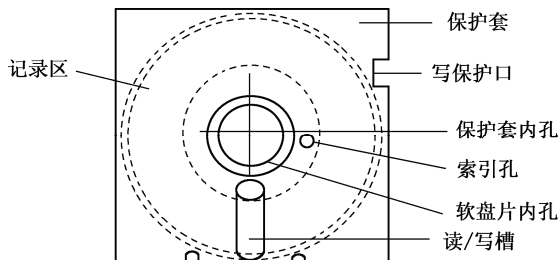


图 4-34 3 英寸软盘外形图

一个区域及外周边缘是空白区。以 3 英寸软盘片为例，记录区宽度约为 2 cm；保护套上开了一个

读写槽，盘片旋转时，磁头通过读写槽对盘片进行读写；在内部空白区中有一小孔（索引孔），盘片每旋转一周产生一个索引脉冲，用光电方法进行检测。

保护套一般由两层构成，外壳多用塑料板制成，内衬用柔软的防静电的特种纸或无纺布制成。因此，软件的保护套可以保护盘片表面免受划伤，同时擦拭盘面使其保持清洁，可防止静电作用引起的数据丢失。保护套中心的大圆孔是起定位作用的装卡孔，允许软盘片套入驱动主轴上。工作时，盘片的夹紧机构夹住保护套使其不动，盘片则在主轴电动机驱动下，在套内自由转动。椭圆形孔即前述读写槽，允许磁头沿径向寻道并与盘片接触，进行读写操作。保护套的索引与盘片索引孔相对应，当盘片索引孔经过保护套索引孔时，光电检测装置产生索引脉冲，既提供磁道起始位置信息，又可由索引脉冲频率检测盘片转速。

保护套边缘有一个写保护缺口，光电检查产生的信号允许写入磁盘。如果封贴该缺口，则光源不能通过，没有允许写入信号，驱动器将只允许读盘而不能写入，软盘上记录的信息不被删改。

（2）分类

最基本的分类标准是按盘片直径分类，常用的有 8 英寸、5.25 英寸、3.5 英寸、2.5 英寸等。最早出现的是 8 英寸软盘，现已较少使用。目前还在应用的是 3.5 英寸盘、2.5 英寸盘，普遍采用改进型调频制，俗称双密度。

按软盘驱动器外形尺寸，又分为标准高、半高、三分之一高驱动器。目前常用的半高软盘驱动器外形尺寸为 203 mm×146 mm×41 mm。

2. 信息分布与软盘寻址信息

记录信息分布在盘片的两个记录面上，每面分为若干磁道，每道又划分为若干扇区。

（1）磁道

读写时，盘片旋转而磁头固定不动。盘片旋转一周，磁头的磁化区域形成一个磁道。在磁道内，逐位串行地顺序记录。每当磁头沿径向移动一定距离，可形成又一磁道。因此记录面上将形成一组同心圆磁道。最外一圈为 0 道，作为磁头定位的基准，往内道号增加。

沿径向，单位距离的磁道数称为道密度。每片容量 1.44 MB 的 3.5 英寸双面高密软盘，每面有 80 道，道密度 135 tpi（每寸磁道数）。每片容量 2.88 MB 的 3.5 英寸双面超高密软盘，每面有 80 道，道密度 270 tpi（每寸磁道数）。新型软盘的道密度甚至还要高出数倍。

（2）扇区

一个磁道沿圆周又划分为若干扇区，每个扇区内可存放一个固定长度的数据块。通常靠磁道内记录格式进行扇区的划分与识别，称为软划分。

一道内的扇区数和扇区内记录容量大小，与操作系统的版本及驱动器容量有关。如微机中常用的 1.44 MB 软盘，每道分为 18 个扇区，每个扇区可存放有效数据为 512 B。2.88 MB 软盘，每道分为 36 个扇区，每扇区 512 B。对于常用软盘，各道容量相同。

沿磁道圆周，单位距离可记录的位数称为位密度。写入电流按一定频率形成位单元，但外圈的线速度大于内圈，因此外圈的位密度小于内圈。通常磁盘给出的是最内圈的位密度，即最大位密度。现在 1.44 MB 软盘的最大位密度超过 1000 bpi，而新型软盘可达 25 kbp 甚至更高。注意，各道位密度不同，而道容量相同。

在分析了软盘中信息分布之后，我们可以归纳出，访问软盘存储器时应当给出台号、磁头号、磁道号、扇区号、交换量等寻址信息。台号即软盘驱动器号，多数微机系统配有两台软盘驱动器，称为 A 盘和 B 盘，相应的台号为 0 和 1。每个记录面上有一个磁头，因此磁头号也就是记录面号。对于双面盘片，分别定义为 0 号磁头（0 面）和 1 号磁头（1 面）。磁道号是磁头定位（寻道）的

依据。扇区号通常指文件的起始扇区号。交换量指文件占有的扇区数。

3. 机械结构

根据软盘驱动器的工作方式，可分为三个方向的机械运动，因此设置了相应的装置。

(1) 主轴驱动装置

现在多由直流电动机直接驱动盘片，作稳速旋转。所写的软盘可以插入其他系统工作，读写时都按标准的转速工作。

(2) 磁头定位装置

为使磁头沿径向寻道，准确地定位于指定磁道之上，需要一个精度很高的磁头定位装置，这是机械结构中的关键部件。由步进电动机驱动磁头小车，小车上安装两个磁头，每个记录面一个。两个磁头同步移动，如果 0 面磁头定位于 03 道，则 1 面磁头也同时定位于该面的 03 道上。寻道时从 00 道开始，每发送一个步进脉冲，磁头移动一个道距，属于开环控制方式。

(3) 加载机械与夹紧机构

使磁头位于工作位置，称为加载。插入软盘并关上小门后，加载继电器通电，加载臂就以其软垫将磁头压向盘片。加载前，磁头与盘片间有一定距离。加载后，磁头与盘片接触，称为接触式。断电后磁头离开盘片。

如前所述，在磁盘的每个记录面，读/写共用一个磁头。在读/写磁头的两侧，装有两个抹磁头，在写入时可抹去两侧的边缘部分（磁化不充分部分），使磁道只留下有效部分，而磁道之间则形成一个空白区，以减少磁道间的相互干扰。

由于磁头采取接触式，驱动器使用时间长了，磁头位置可能发生变化，影响正常读写。可以采用一种 CE 盘（用户工程师盘，俗称猫眼盘），帮助调整磁头位置（径向位置与方位角）。CE 盘又称为校准盘，在它的某些道上预先记录了一些特殊信息，校准时可由示波器观察相应的读出信号，判断磁头位置是否正确。但调整次数多了会使磁头的固定机构松动，因此应当谨慎。

4. 磁道记录格式

某一时刻，磁盘只有一个记录面上的一个磁头工作，即只有一个磁道被读或写，道内逐位地串行记录，可称为单道串行记录。在一个闭合磁道内，信息的组织称为磁道记录格式，简称为磁道格式。磁道记录格式与操作系统的版本有关，磁盘控制器与之配套。为了使记录盘片能够通用，一种计算机系列中往往采用某种标准磁道格式。但有的操作系统还允许用户产生自己的非标准物理格式，防止他人非法复制。

图 4-35 是在微机中广泛使用的 IBM34 系列磁道格式。一个磁道被软划分为 9 个扇区或 15 个扇区，每个扇区又分为标志区与数据区，各自包含若干项。记录的有效数据或程序，位于数据区的 DATA 中。其他信息则是为了识别有效数据而设置的格式信息。

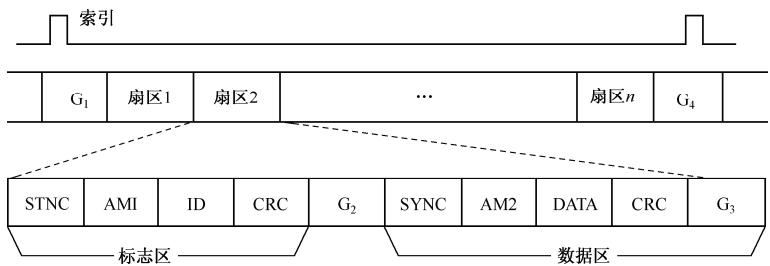


图 4-35 IBM34 系列磁道格式

① 索引标志。盘片旋转一周，索引孔通过光电检测产生一个索引脉冲，经整形后，脉冲前沿标志着一个磁道的开始。如果没有索引脉冲，则一个闭合圆环的磁道将无法区分其头尾。

② 首部。间隔 G_1 ，即磁道的起始部分，表明采用何种记录方式，为 16 字节 FF (FM) 或 16 字节 4E (M^2F) 记录方式。在 G_1 之后，将开始第一个扇区。

③ 标志区。每个扇区的开头是一个标志区，提供该扇区的标志信息 ID。但为识别 ID，又需增设一些格式信息。因此标志区共包含：

- ◎ 同步字段 SYNC，6 字节 00。

- ◎ 标志区地址标志 AM1，1 字节 FE (7 个 1 之后一个 0，表示后面将是标志信息)。

- ◎ 标志信息 ID，4 字节，分别给出道号 C、磁头号 H、扇区号 R、扇区长度 N (因此寻道后读出相应 ID 段，可知寻道是否正确。道内读写时，可由 ID 获得扇区号，作为寻找扇区依据。IBM34 系列磁道格式允许每个扇区有效数据长度为 128 B、256 B、512 B、1 KB 等规格，扇区长度 N 以编号形式指明是哪一种长度规格)。

- ◎ CRC，2 字节，标志区循环校验码 (即余数)。

④ 间隔 G_2 ，作为标志区与数据区之间的间隔，为 11 字节 FF (FM) 或 11 字节 4E (M^2F)。

⑤ 数据区，包含：

- ◎ 同步字段 SYNC，6 字节 00。

- ◎ 数据区地址标志 AM2，1 字节 F8，表示后面是有效数据。

- ◎ 数据 DATA，存放有效记录数据。如前所述，它有几种长度规格，常用的是 512 B。

- ◎ CRC，2 字节，数据区循环校验码。

⑥ 间隔 G_3 ，表示一个扇区的结束，为 27~116 字节 FF (FM) 或 27~116 字节 4E (M^2F)。

⑦ 间隔 G_4 ，表示一个磁道的结束。

4.5.2 硬盘存储器

硬盘存储器的工作原理与软盘存储器基本相同，但作为外存储器的主体，它追求更大的存储容量、更长的使用寿命，以及更高的数据传输率。因此，要求采用更多的硬质记录盘片、更高的记录密度、浮动式磁头、高精度定位系统，甚至是密封的整体结构。相应地，造价也较软盘存储器为高。本节将着重介绍其与软盘存储器不同的部分。

1. 盘片、盘组、磁盘阵列

(1) 盘片

为了提高记录密度，并适应高度精密的驱动器结构，采用硬质基体制成非常平整光滑的圆形盘片，再在盘片上生成一层很薄且很均匀的记录磁层，所以称为硬盘。常规硬盘采用铝合金基片，上面电镀磁层。新型硬盘则采用光洁度更高的工程塑料、玻璃、陶瓷作基片，用溅射工艺形成更薄、更均匀的记录磁层。

最早的硬盘片外径为 24 英寸，后来缩小为 14 英寸、8 英寸、5.25 英寸、3.5 英寸、2.5 英寸、1.8 英寸、1.3 英寸甚至更小的微型硬盘。

由于制造工艺的提高，盘径尺寸不断减小，使盘片的转动惯量减小，容量却不断增大，使用就更广泛。

(2) 盘组

为了提供更大的存储容量，将多个盘片组装在同一主轴上，可同轴旋转，称为盘组。为了提

高精度与传动稳定性，多将盘组的主轴与驱动电机（直流无刷电机）的转子做成一体，即将各盘片直接装在主轴电动机的转轴之上，盘片之间用间隔区分，间隔约为 10~20 mm，如图 4-36 所示。

微机系统中的小型硬盘驱动器一般采用 2 片、4 个记录面。大容量硬盘驱动器中可采用多片，如 6 片、12 片一组。相应地，将一个驱动器的容量称为每轴多少字节。

根据盘组是否可拆卸，硬盘存储器常分为可换盘片式和固定盘片式两种。

可换盘片式硬盘驱动器的盘组与主轴电动机的转轴分离，盘组做成圆盒形，可整体拆卸以脱机保存，可更换装入新的盘组。

固定盘片式的盘组与主轴电动机的转轴不可分离更换，且常采取密封结构。在中、小、微型硬盘驱动器中普遍采取固定盘片式结构。

（3）磁盘阵列

磁盘阵列（Redundant Arrays of Independent Disks, RAID），是由美国加州大学伯克利分校的 D.A.Patterson 教授在 1987 年提出的。可以把 RAID 理解成一种使用磁盘驱动器的方法，它将一组磁盘驱动器用某种逻辑方式联系起来，作为逻辑上的一个磁盘驱动器来使用。实质上它是由很多价格较便宜的磁盘，组合成一个容量巨大的磁盘组，并用一个盘阵列控制器进行控制，利用个别磁盘提供数据所产生加成效果提升整个磁盘系统效能。利用这项技术，将数据切割成许多区段，分别存放在各个硬盘上，如图 4-37 所示。

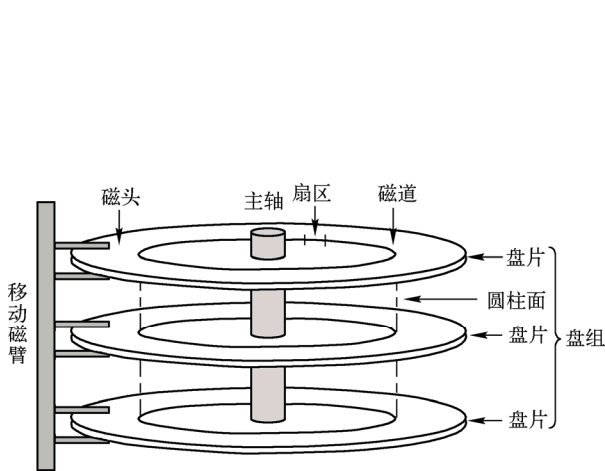


图 4-36 盘组的结构示意

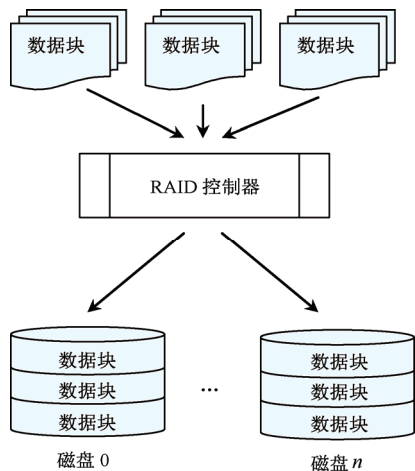


图 4-37 磁盘阵列示意

磁盘阵列还能利用同位检查（Parity Check）的观念，在数组中任一硬盘故障时，仍可读出数据，在数据重构时，将数据经计算后重新置入新硬盘中。一般情况下，组成的逻辑磁盘驱动器的容量要小于各个磁盘驱动器容量的总和。RAID 的具体实现可以靠硬件也可以靠软件，Windows NT 操作系统就提供软件 RAID 功能。

磁盘阵列所用的单台驱动器容量不是很大，比较容易制造，而整个盘阵列的容量可以做得很大，且数据传输率高。盘阵列还可做成容错结构，当一个或数个驱动器发生故障时该盘阵列仍可以正常工作，因此可靠性较高。由于这些特点，磁盘阵列技术受到重视，发展前途很光明。

磁盘阵列其样式主要有三种：外接式磁盘阵列柜，内接式磁盘阵列卡，利用软件来仿真。外接式磁盘阵列柜最常被使用大型服务器上，具可热抽换（Hot Swap）的特性，不过这类产品的价格都很贵。内接式磁盘阵列卡，因为价格便宜，但需要较高的安装技术，适合技术人员使用操作。利用软件仿真的方式，由于会拖累机器的速度，不适合大数据流量的服务器。

RAID 包括多种实现方案, 如: RAID0 (基本模式), RAID1 (磁盘镜像), RAID 0+1 (LSI MegaRAID (数据保护)、Nytro (数据加速) 和 Syncro (数据共享)), RAID2 (带海明校验), RAID3 (奇偶校验且并行传输), RAID4 (奇偶校验且独立磁盘结构), RAID5 (分布式奇偶校验且独立磁盘结构), RAID6 (带有两种分布存储、奇偶校验码、独立磁盘结构), RAID7 (优化的高速数据传输磁盘结构), RAID10 (高可靠性与高效磁盘结构), RAID53 (高效数据传输磁盘结构), 等等。

RAID 的优点如下:

① 成本低, 功耗小, 传输速率高。在 RAID 中, 可以让很多磁盘驱动器同时传输数据, 这些磁盘驱动器在逻辑上又是一个磁盘驱动器, 所以使用 RAID 可以达到单个的磁盘驱动器几倍、几十倍甚至上百倍的速率。

② 提供容错功能。这是使用 RAID 的第二个原因, 因为普通磁盘驱动器无法提供容错功能, 如果不包括写在磁盘上的 CRC (循环冗余校验) 码。RAID 的容错是建立在每个磁盘驱动器的硬件容错功能之上的, 所以它提供更高的安全性。

③ RAID 比起传统的大直径磁盘驱动器来, 相同容量下, 价格要低很多。

2. 温彻斯特技术

1956 年, IBM 公司制成第一台硬盘存储器 IBM350, 盘片外径 24 英寸, 容量 5 MB, 平均寻道时间长达 600 ms。此后, 硬盘技术经历了四代的更新发展。第一代硬盘的主要技术特征是: 用 24 英寸盘片, 盘组可达 50 片, 固定盘片不可卸, 通过外供压缩空气使磁头浮动于盘面之上。第二代特征是: 采用 14 英寸盘片, 盘组可拆卸更换, 依靠气垫原理使磁头浮动。第三代特征是磁头定位装置采用音圈电动机驱动与伺服盘定位技术。第四代硬盘技术称为温彻斯特技术, 采用这种技术制作的硬盘常被简称为温盘。

第四代硬盘产品首先是由 IBM 公司的一个研究所研制成功的。这个研究所位于美国加州坎贝尔市温彻斯特 (Winchester) 大街, 人们将这种以他们为代表的第四代硬盘技术称为温彻斯特技术, 于 1973 年首先用于 IBM3340 硬盘系统中。温彻斯特技术实际上是一系列综合改进的制造技术, 它的主要特点有以下四方面。

(1) 密封的头、盘组件

温彻斯特技术最主要的特征是将磁头、盘组、定位机构、主轴电机等主要部件密封在一个盒中。这就消除了可卸盘机械结构不够稳定等因素, 提高了定位精度。因此, 温彻斯特盘属于固定盘结构。密封是靠磁性流体密封技术实现的。将极细的铁钴微粉粒与分散液相混合, 填充于盘腔与外部的连接处。在磁场作用下, 磁性流体使盘腔密封, 腔内净化高达 100 级, 即 $0.5\text{ }\mu\text{m}$ 以上的尘埃数少于 3.5 个/千克 (人的头发粗约 $80\text{ }\mu\text{m}$, 一般尘埃直径约 $40\text{ }\mu\text{m}$, 烟雾中的微粒直径约为 $5\text{ }\mu\text{m}$)。可见, 硬盘驱动器的密封腔内净化程度很高, 但对外部环境则要求不高。在一般环境下切勿打开密封腔。

密封室的基座与上盖选用导热性好的金属材料制成, 复杂的硬盘驱动器在密封室内还设置有自循环散热系统。

(2) 采用薄膜磁头、溅射薄膜磁层

采用微型薄膜磁头后, 磁头尺寸减小、重量减轻, 使浮动高度降低。与此配套的薄膜磁层, 是用集成电路制造工艺中的溅射工艺生成的, 厚度薄、内部结构均匀、表面光洁度高。这些措施有利于记录密度的提高。

(3) 将集成化的读写电路安置在靠近磁头的位置, 以改善高频传输特性、减少干扰。

(4) 接触起停式浮动磁头

硬盘不允许在读写时磁头接触记录区，否则会划伤磁道。这是因为硬盘与软盘不同，采用硬质基体，转速也高。因此，硬盘在工作时要求磁头浮空，离开盘面。

通常在记录区以内（靠近轴心）有一空白区，被当成“起停区”（又称为着陆区），也有的硬盘将起停区设置在记录区往外的空白区上。不工作时（如未上电时），盘片不旋转，磁头停在起停区内，并与盘面相接触。

驱动器加电后，盘片开始旋转并加速到额定转速。当达到一定转速时，由于盘片旋转所造成的气流形成气垫，气垫的浮力使磁头浮起，离开盘面，称为磁头从起停区起飞。盘片达到额定转速需时约 6~10 s，在额定转速下磁头浮动高度仅数分之一微米。然后定位机构使磁头从起停区移至记录区，并定位于 00 磁道，称为“定标”。此时可令驱动器进行寻道及读/写操作。

3. 磁头定位系统

由于硬盘的道密度高，而且要求快速寻道，因此对磁头定位机构的要求比软盘更高，这是驱动器中的关键部件。定位系统包括运载部件、驱动部件、控制部件等。运载部件又称磁头小车，各记录面的磁头（每面一个）装在小车的读写臂上，呈梳状，同步地沿径向移动，同时指向各自记录面的相同序号磁道上。驱动电动机与相应的控制系统有以下两大类。

（1）步进电动机驱动、开环控制

在小容量硬盘驱动器中，如容量 20 MB 的 5.25 英寸温盘，其道密度约为 300 tpi，就采用步进电动机驱动定位机构，相应地设置现行道地址计数器 CAC 和目标道地址寄存器 CAR。启动时先让磁头定位于 00 道，CAC=0，求出二者差值即所需的步进脉冲数。

（2）音圈电动机驱动、闭环控制

在大、中容量的硬盘驱动器中，多采用音圈电动机驱动定位机构。通过速度控制以提高寻道速度，通过位置控制以提高定位精度。

音圈电动机的工作原理类似于电动式喇叭的音圈运动原理，并由此得名。如图 4-38 所示，动圈位于永磁磁路之中，当驱动电流注入动圈时，电磁力推动动圈运动，从而带动梳形读写臂在盘片上作径向运动。音圈电动机分为两种：直线形，又称直线电动机，可直接驱动磁头作直线运动（如图 4-38 所示），减少了常规旋转电动机因传动换向机构带来的传动误差，因而是常用的一种；旋转型音圈电动机，在某些结构中需作弧形运动，因而需将音圈电动机做成旋转型。

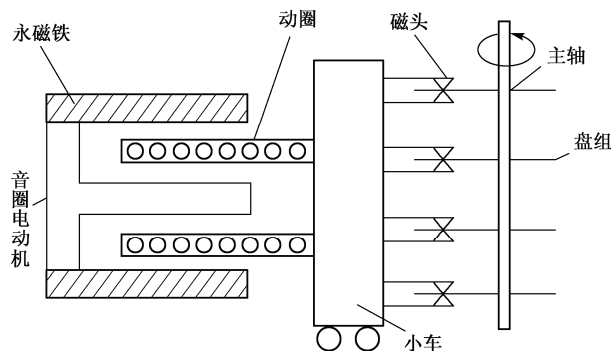


图 4-38 音圈电动机与定位机构示意图

对音圈电动机的反馈闭环控制系统分为两级，常用单片机或专门的信号处理控制器 DSP 进行控制。

① 速度控制（粗控阶段）。定位机构启动后，经过加速达到额定速度，以尽可能快的速度移

向目标磁道；在接近目标磁道时，转入减速刹车阶段。为了停在目标磁道上，减速刹车阶段的反馈调节是关键。如果寻道距离较短，定位系统可能还未达到额定速度就已转入减速阶段，即只有加速段与减速段。

② 位置控制（精控阶段）。准确定位对于读写的可靠性至关重要。当道密度提高、磁道宽度变小后，步进电动机开环控制方式不能满足要求，因而采用音圈电动机闭环调节。位置控制的关键是提供精确的位置检测信号，作为反馈调节的依据。早期曾用过光栅式、感应同步器等位置检测手段，在高精度硬盘定位系统中则采用伺服技术进行检测。后者又分为以下 3 种。

● 伺服盘方式

在盘组中选取一个记录面，专门用来记录磁道位置信息，称为伺服盘面。伺服面上的磁道数与数据面上的磁道数相同，我们将伺服面上的磁道称为伺服道，将数据面上的磁道称为数据道。如前所述，伺服面上的磁头与各数据面上的磁头同步移动，伺服道中心位置与数据道中心位置相差半个道距。换句话说，数据道中心正好处于两个伺服道之间。

伺服面的奇数道与偶数道，分别记录两种不同的特殊编码。一种方法是：让伺服面磁头读出信号 e 与磁道位置间的关系呈三角波波形，如图 4-39 所示。当 $e = 0$ 时，磁头位于两个伺服道之间，也就是位于数据道的中心。据此可判断磁头定位的位置，统计 e 过零次数可以知道磁头所越过的磁道数。

● 嵌入式（分段式）伺服方式

伺服盘方式占用一个记录面，减少了盘组的有效记录面积。伺服面与各记录面虽然组装于同一个盘组之中，但毕竟分属不同盘面，机械位置变动和温度影响等因素，还是会在一定程度上影响定位精度。为了克服这些缺点，又出现了嵌入式即分段方式。

嵌入式伺服方式是在数据面上安排一段定位伺服信息，不另设专门的伺服面，如图 4-40(a) 所示。通常是在闭合磁道的末尾，即索引脉冲到来之前约 $200\ \mu\text{s}$ 的一段区域，安排成伺服区，又称为索引伺服区。因此，这种嵌入式伺服方式又称为索引伺服方式。由于它紧靠数据区，温度影响极相近，又共用同一个磁头，所以定位精度更高；缺点是要等待索引伺服区到来时才能获得位置信号，使定位时间加长。

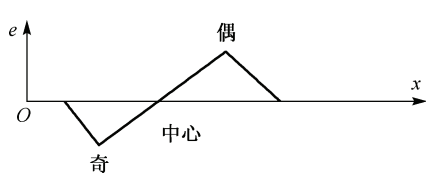


图 4-39 伺服头读出波形

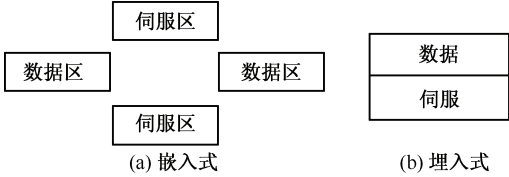


图 4-40 伺服方式示意图

另一种方法是在每个扇区的开始区域嵌入伺服道，则定位时间减小，而且温度影响更小。这种嵌入式又称为扇区伺服方式。

● 埋入式伺服方式

埋入式伺服方式是将磁层分为上下两层，上层记录数据，下层记录磁道伺服信息，如图 6-35(b) 所示。其效果更好，但技术难度加大。

有的磁盘驱动器采用了双重伺服技术，将伺服盘方式用于速度控制粗定位，将扇区伺服方式用于位置控制精定位。其相应的控制机构十分复杂。

4. 硬盘信息分布与寻址信息

(1) 磁道

与软盘相同，硬盘的每个记录面划分为若干呈同心圆环的磁道，只是道密度更高，依靠索引脉冲标志一个磁道的起始位置。

(2) 圆柱面

一个盘组有若干记录面，所有记录面上相同序号的磁道构成一个圆柱面，圆柱面号与道号相同，如 00#圆柱面、79#圆柱面等。每面的道数即盘组的圆柱面数。

为什么要引入圆柱面这一概念？让我们考虑这样一个问题：磁盘上的信息通常以文件的形式组织并存储，如果一个磁道存放不完，我们将它继续存放在同一记录面的相邻磁道上？还是将它继续存放于同一圆柱面的相邻记录面上？如果采用第一种方法，则更换磁道时必须进行寻道操作，所需时间较长。如果采取后一种方法，由于定位机构使所有记录面的磁头都对准同一序号磁道，大家都处于同一圆柱面中，只需通过译码电路选取相邻面的磁头，即可继续读写，几乎没有时间延迟。所以引入圆柱面这一级，让文件尽可能存储于同一圆柱面上，然后才是相邻圆柱面。

(3) 数据块

每个磁道可存放若干数据块，可分为定长数据块与不定长数据块两类记录格式。

在小容量硬盘中，大多采用定长数据块格式。相应地，每个磁道划分为若干扇区，每个扇区中存放一个定长数据块，与软盘相似。

在大容量硬盘中，有的采用不定长数据块格式，该长则长，能短则短。相应地，不采取扇区划分（扇区一般只用于等长度划分格式），直接以数据块号或记录号标志。

因此，硬盘寻址信息一般由驱动器号、圆柱面号（道号）、磁头号（记录面号）、数据块号（记录号，对于定长数据块格式用扇区号）和数据交换量等组成。

5. 磁道记录格式

(1) 定长数据块磁道格式举例

图 4-41 是在 IBM 系列微机硬盘中常用的定长数据块格式，与软盘格式相似。

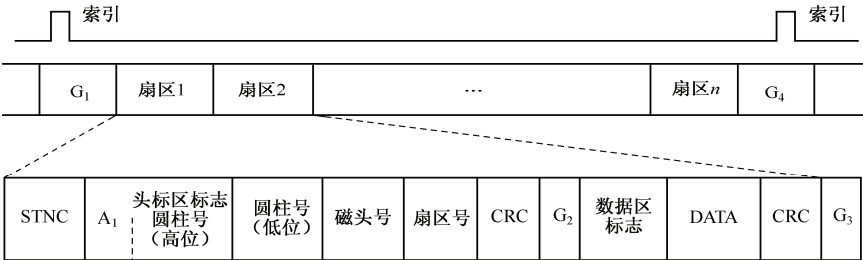


图 4-41 磁盘磁道格式（定长的数据块）

索引脉冲的前沿标志着磁道的开始。间隔 G₁，16 字节的 4E。每个扇区包含下述 5 部分：

- ① 同步区 SYNC，14 字节 00。经历这一段，使控制器中的锁相电路与读出信号序列的频率同步，为读头标区做好准备。
- ② 头标区（地址标识区），共 7 字节。
 - ⊙ 头标区标志，其中第 1 字节为 A₁，第 2 字节给出圆柱号高位。
 - ⊙ 圆柱号低位，1 字节。
 - ⊙ 磁头号，1 字节。其中第 0、1、2 位为磁头号，可标志 8 种头号；第 3、4 位为 0；第 5、

6 位为扇区计数值；第 7 位为坏块标志，若该位为 1 表示本扇区段为坏块，不能使用。

- ⊙ 扇区号，1 字节，存放逻辑扇区号。
- ⊙ CRC 校验，2 字节，存放头标区的循环校验码。
- ③ 间隔 G_2 ，16 字节的 00。
- ④ 数据区
- ⊙ 数据区标志，2 字节：A1F8。
- ⊙ 数据 DATA，定长数据块，有 128 B、256 B、512 B、1024 B 等几种规格。常用的 PC DOS 操作系统多采用 512 B 规格。
- ⊙ CRC，2 字节。
- ⑤ 间隔 G_3 ，3 字节 00 及 15 字节 4E。

(2) 不定长数据块磁道格式举例

IBM 3350、3357、3380 等磁盘采用不定长数据块磁道格式，如图 4-42 所示。一个磁道中的信息由若干记录 (Record) 组成，每个记录包含计数段 (Count)、关键字段 (Key)、数据段 (Data) 三段，所以这类磁道格式又被称为 CKD 结构。

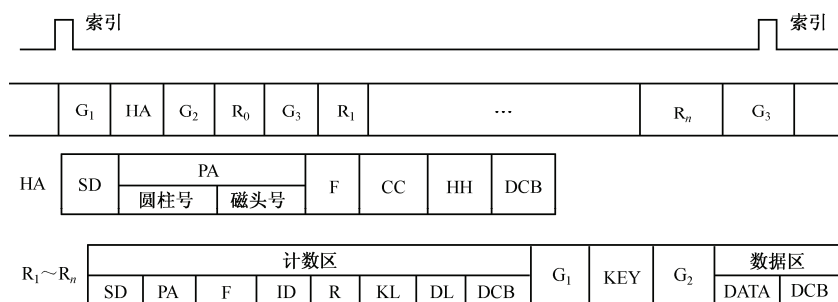


图 4-42 硬盘磁道格式 (不定长的数据块)

标准的 CKD 布局磁道格式大致如下：索引脉冲标志着磁道的开始。间隔 G_1 ，共 116 字节，其中 115 字节为 00，另一字节为 19 (同步信息)。

- ① 标志地址 HA (Home Address)。这是用来标志磁道的地址区，共 20 字节，其中：
 - ⊙ SD (Skip Displacement)，6 字节，用来指出该磁道上有几个瑕疵，以及它们在磁道上的位置。
 - ⊙ PA (Physical Address)，3 字节，表明磁道的物理地址。其中，第 1、2 字节为圆柱面地址，第 3 字节为磁头号。
 - ⊙ F (Flag)，1 字节，用来说明磁道状况，如该磁道是好的磁道或是坏的磁道，是基本磁道 (又称为原始道) 还是替补 (备用) 磁道等。
 - ⊙ CC (Cylinder Address)，圆柱面逻辑地址，2 字节。
 - ⊙ HH (Head Address)，2 字节，磁头逻辑地址。CC 与 HH 的含义是：若本磁道是好的原始道，则逻辑地址就是该道磁头的物理地址；若本磁道是坏道，已用备用磁道替代，则逻辑地址就是替补道的物理地址。
 - ⊙ DCB (Detection Code Byte)，6 字节，错误校验字节。

间隔 G_2 ，76 字节，其中 75 字节 00，1 字节 19。 G_2 用来分隔 HA 与 R_0 。

② 零号记录 R_0

在磁道标志地址 HA 之后，经 G_2 分隔，首先是 R_0 ，它不是用户的数据记录，而是用来说明

本磁道的有关状态的。如果磁道发生故障，将依据 R_0 提供的信息实现磁道替换。与一般用户记录区不同， R_0 只应用计数区部分，不设置关键字段；虽然设置了数据区但一般不用。对计数区组成，后面将要介绍，所以对 R_0 不专门细述。

③ 用户记录 R_i

在各记录之间用 G_3 分隔， G_3 为 79 字节，其中 65 字节 00、3 字节地址标记、10 字节 00、1 字节 19。

● 计数区 Count，包含：

⊙ SD，6 字节，同前（指本区有瑕疵）。

⊙ PA，3 字节，同前。

⊙ F，1 字节，同前。

⊙ ID 识别区，提供 CCHH 信息，5 字节：

| 磁道状况 | CCHH |
|-------|------------|
| 好的原始道 | 本道的物理地址 |
| 坏的原始道 | 替补道的物理地址 |
| 好的替补道 | 所替换的坏道物理地址 |
| 坏的替补道 | 本道的物理地址 |

⊙ R，1 字节，记录号。

⊙ KL，1 字节，本记录的关键字长度。

⊙ DL，2 字节，本记录内数据长度。

⊙ DCB，6 字节，校验码。

● 关键字区 KEY：关键字是数据的识别信息，如顺序号、密码等。关键字区也有自己的校验码字节 DCB。

● 数据区 DATA：用来记录用户数据，允许使用不同长度，然后是 6 字节的数据校验码。

在一个记录内的三个区之间，也用 G_2 分隔。

如上所述，在进行硬盘格式化时，可以发现坏道并将其更换。用户使用的逻辑地址可以不变，磁道格式信息 CCHH 提供替换道的物理地址。

6. 硬盘存储器的发展方向

20 世纪 90 年代多媒体计算机的出现，对硬盘驱动器提出了更高的要求。为了满足存储视频、语音和音乐等信息的需要，以及满足 Windows 操作系统和便携机的需要，使硬盘驱动器加速向小型、轻量化、大容量、快速低功耗和低价格方向发展。

硬盘（Hard Disc Drive，HDD）的发展已经历了 50 多年。1956 年，IBM 公司首创的世界第一台 HDD IBM 350，面记录密度仅为 2000 位/平方英寸，硬驱容量仅为几百 KB；2004 年，IBM 公司为笔记本电脑装配的 2.5 英寸硬盘，其容量为 80 GB，现在已有高达几 TB 的硬盘了。

磁盘驱动器主要采取了以下技术措施来提高其性能：

① 采用浮动磁头，降低浮动高度。浮动高度已从 $20\text{ }\mu\text{m}$ 降低到 $0.1\text{ }\mu\text{m}$ 以下。

② 采用伺服机构，提高磁道密度。从最初的磁道密度 20 tpi，提高到 5000~17000 tpi。一般采用嵌入伺服或光伺服技术。

③ 采用温彻斯特技术。

④ 采用 MR 磁头和 PRML 信号处理方法。可以确保提高读出信号幅度，提高信号噪声 (S/N) 比，使数据传输率和面记录密度分别提高了 30%~40% 和 30%~50%。

⑤ 改进存储介质和盘基，提高记录密度。改进存储介质材料性质和制造工艺，采用垂直磁

记录方式，提高记录密度。

⑥ 缩短平均存取时间，提高数据传输率。主要措施包括缩短磁头行程，提高主轴转速（从 3600 rpm→4500 rpm→5400 rpm→6300 rpm→7200 rpm→15000 rpm），使平均寻找时间小于 10 ms，平均等待时间缩短为 4.2 ms。扩大超高速缓冲存储器容量（>8 MB），改进接口，发展廉价冗余磁盘阵列（RAID）等。有的磁盘采用恒密度记录，即从内径开始，随着半径的增加，相应磁道的存储量增加但保持位密度不变。

⑦ 硬盘的微小型化。磁盘直径经历了 14 英寸→8 英寸→5.25 英寸→3.5 英寸→2.5 英寸→1.8 英寸，甚至到 1.3 英寸。设备厚度经历了 82 mm→41 mm→25.4 mm→17 mm。

磁盘驱动器的容量大致以每年翻一番的速度增长。主要措施是提高面记录密度和增加内装盘片数量，但增加盘片数量后会导致主轴电动机功耗的增加、设备加厚。目前在工作站和 PC 机中使用最多的是 3.5 英寸的驱动器，为了配合多媒体对硬盘的大容量要求，早在 2004 年已有容量大于 160 GB 的硬盘面市，传输速率已可达 3 Gbps。

早期的硬盘采用 IDE（Integrated Drive Electronics）数据传输接口，数据线并行 40 针，电源接口 4 针，直流 5V 驱动电压，这种接口也被称为并行 ATA（Paralell Advanced Technology Attachment）接口。ATA 接口标准从 ATA-1 一直发展到 ATA-7（即 ATA133），与此相应的硬盘其数据传输速度可达 133MB/s。发展到 ATA-7 阶段后，由于 ATA 硬盘的并行接口的电缆属性、连接器和信号协议都遇到了很大的技术瓶颈，还存在不支持热插拔、冗错性差、功耗高、影响散热及连接线长度短等缺陷。

要在技术上突破这些有很大难度，于是 ATA 硬盘退出了历史舞台，逐渐被另一种新型接口标准的硬盘取代。

将 ATA 接口的硬盘完全取代的是一种串行接口硬盘（Serial ATA，SATA），这种硬盘采用并行数据传输接口，其电源接口是 15 针（每 3 针为一组，共 5 组），数据接口仅 7 针，直流 0.5 V 驱动电压，如图 4-43 所示。

由于技术的原因，第一代 SATA 硬盘（SATA 1.0 标准），其数据传输速度也仅能达到 150MB/s，传输距离为 1 米。随着技术的不断发展，历经 SATA 2.0 阶段后，如今已发展到

SATA 3.0，此时的 SATA 硬盘其数据传输速度已可高达 600MB/s，比第一代硬盘提高了 4 倍，其最大传输距离也达到了 2 米。

与早期的 ATA 硬盘相比，SATA 硬盘具有发热量小、功耗低、传输距离更远、传输更可靠、接口电路结构更简单、支持热插拔等优点。

除了普通硬盘（ATA 或 SATA）外，硬盘大家族还有一种基于半导体技术的硬盘，俗称固态硬盘（Solid State Drives，SSD）。这种硬盘的存储介质是 FLASH 芯片或 DRAM 芯片，外形尺寸与普通硬盘无异，其接口除了支持经典的 SATA 标准外，一般还支持 MSATA 和 CFast 接口。第一款固态硬盘诞生于 1989 年，发展到现阶段，固态硬盘除了用于笔记本硬盘外，更多是用于移动存储设备（如移动硬盘和 U 盘等）。在工业界，近年甚至出现了 SSD+HDD 的混合型硬盘，这种硬盘将 SSD 的高速特性与 HDD 的大容量特性完美结合，可以使得硬盘的整体性能接近固态硬盘，又能提供更大的存储空间。现阶段，无论是普通固态硬盘，还是混合型硬盘，都尚未大规模替代普通的磁盘作为主存的直接后援。或许，随着技术的不断发展，在不久的将来固态硬盘将会全面替代现在主流的普通磁盘。



图 4-43 SATA 和 PATA 硬盘接口

4.5.3 技术指标与数据校验

1. 磁盘的格式化

由于不同计算机系统的磁道格式可能不同,所以盘片在出厂时并没有写入上述格式信息。因此,空白盘片在使用前需要进行格式化,即用操作命令按其格式写入格式信息,进行扇区划分等,然后才能写入有效程序与数据。当用户的软盘在使用中发现有故障或沾染上病毒时,也可通过格式化清除原有信息,重新记录。

① 建立磁道记录格式,称为物理格式化或初级格式化。经过这一级格式化,磁道被软划分为若干扇区,每个扇区又划分为标志区与数据区,并写入同步字段 SYNC、地址标志 AM1 与 AM2、标志信息 ID 等,但 DATA 段空着,待写入信息。

② 建立文件目录表、磁盘扇区分配表、磁盘参数表等,这些称为逻辑格式化或高级格式化。有关内容将在“操作系统”课程中介绍。

2. 磁盘存储器技术指标

软盘和硬盘的技术指标项目基本相同,在这里一并描述。

(1) 磁盘容量

存储容量是磁盘的一项重要技术指标,有以下两种容量指标。

① 非格式化容量

非格式化容量=面数×道数/面×内圈周长×最大位密度

这一指标表明一个磁盘所能存储的总位数,其中包括各种格式信息、间隔信息等。由于磁盘各道安排的存储容量完全相同,通常只给出最内圈磁道的位密度(即最大位密度),所以我们选取最内圈磁道计算道容量。

② 格式化容量

格式化容量 = 面数×道数/面×扇区数/道×字节数/扇区

对用户来说,感兴趣的是在除去各种格式信息后可用的有效容量,即各个 DATA 区容量的总和,称为格式化容量,大约占非格式化容量的 90%左右,如西部数据出品的一款硬盘 WD5000AAKX-083CA1,其标称容量为 500GB,但其格式化容量仅约为 465GB。下面以两种常用软盘驱动器为例,说明格式化容量计算及相应的信息分布情况。

5.25 inch 高密盘(双面双密度磁道)的格式化容量=2 面×80 道/面×15 扇区/道×512 B/扇区=1.2 MB

3.5 inch 高密盘的格式化容量=2 面×80 道/面×18 扇区/道×512 B/扇区=1.44 MB

【例 4-10】某型磁盘由有 3 个盘片组成,共 4 个可用记录面,转速为 7200r/min,盘面的有效记录区其外直径为 356mm,内直径为 100mm,内层记录的位密度为 250b/mm,磁道密度为 8 道/mm,每磁道分 16 个扇区,每扇区 512 字节,计算该磁盘的格式化容量。

格式化容量=4×(356-100)/2×8×16×512B=32 MB

注意:例题中给出的转速为干扰项,而给出的位密度也仅在计算非格式化容量时才需要使用,此外盘面上的磁道称为圆周形,故只能根据有效区半径来计算每面的磁道数。

(2) 工作速度

与随机存储器不同,磁盘的存取时间与信息所在磁道、扇区的位置有关。所以,需要分阶段地描述其速度指标,无法采取单一的速度指标,这是由磁盘工作方式决定的。

① 平均寻道(定位)时间。启动磁盘后,第一步将是寻道,即将磁头直接定位于目的磁道上。每次启动后,磁头首先定标于 00 磁道,以该道为基准开始寻道。如果目的道就是 0 道,则

磁头不需移动；如果目的道是最内圈，则所需寻道时间最长，因此，寻道时间是一个不定值。我们用平均寻道时间（或称为平均定位时间）指标衡量磁盘驱动器的寻道速度。

② 平均旋转延迟（等待）时间。寻道完成后，需在磁道内顺序查找起始扇区，此时磁头不动而盘片旋转。如果寻道完成后起始扇区即将通过磁头下方，则不需要等待；如果寻道完成后起始扇区刚刚通过了磁头下方，则需等待盘片旋转一周（起始扇区再次经过时）才能开始读写。这也是一个不定值，最长等待时间是盘片旋转一周的时间。我们用平均旋转延迟时间（或称为平均旋转等待时间）指标来衡量磁盘驱动器寻找扇区的速度，取决于主轴转速。早期常用 5.25 英寸软盘的主轴转速为 300 转/分，相应的平均旋转延迟为 100 ms。常用 3.5 英寸软驱的主轴转速为 600 转/分，平均旋转延迟时间为 50 ms。由于软盘采用接触式读/写方式，盘片转速过高会加大磁头与盘片之间的磨损，因此限制了主轴转速的提高。

③ 数据传输率（带宽）。找到扇区后，磁头开始连续地读/写：从主存储器中获得数据，写入磁盘；或从磁盘中读得数据，送往主存储器。因此，我们用数据传输率指标来衡量磁盘驱动器的读/写速度，以波特（位/秒）为单位，其物理含义就是单位时间内磁盘输入/输出的数据量。例如，5.25 英寸软驱的数据传输率约为 250 Kbps，3.5 英寸软驱为 500 Kbps。

【例 4-11】 某计算机字长为 32 位，CPU 主频为 500 MHz，磁盘共有 16 个盘面，1024 个圆柱面，每磁道包含 256 个扇区，每个扇区 512 字节，该磁盘旋转速度为 9600 RPM。

（1）计算该磁盘总容量。

（2）计算该磁盘的带宽。

磁盘容量=16×1024×256×512 B=2 GB

磁道容量=256×512 B=128 KB

磁盘转速=9600/60=160 转/秒

磁盘带宽=160×128 KB=20 MBPS

上述 3 个指标分别反映了 3 种工作阶段的速度。用户启动磁盘后，等待多久才能开始获得数据，取决于前两项指标。此时允许 CPU 继续访问主存，执行自己的程序。开始连续读/写后，需要占用总线与主存进行数据传送，将影响主机工作。已知主存的工作速度（存取周期）和磁盘的数据传输率，可以计算出主存需以多大的时间比例，用于向磁盘传送数据，还剩下多少时间比例供 CPU 访存。

3. 磁盘的数据校验

磁表面存储器在磁头与记录介质相对运动中读出，读出信号很小（微伏级），作为外围设备，与主存间的传输距离较长，受干扰较大，因此出错的概率较主存高，相应地需要采取更复杂的校验方法，以判定从磁盘中读取的数据是否正确。

早期，磁表面存储器广泛使用海明校验（Hamming Check, HC）方法来校验磁盘存储的数据，或者对数据进行纠错。现阶段则普遍使用循环冗余校验（Cyclic Redundancy Check, CRC）方法来校验和磁盘数据纠错。CRC 方法也是一种通过软件实现在通信领域中被广泛使用的数据校验和纠错方法。海明校验和循环冗余校验两者都可以用硬件电路逻辑来实现，在磁表面存储器的数据校验中，如果数据的位数较多，则采用循环冗余校验比采用海明校验能更有效地降低硬件成本。对磁盘数据校验时，国际电报电话咨询委员会（CCITT）为 CRC 推荐 $G(x) = x^{16} + x^{15} + x^2 + 1$ ，美国电气和电子工程师学会（IEEE）为其推荐 $G(x) = x^{16} + x^{12} + x^5 + 1$ 。

磁盘每个扇区的容量一般为 512 字节，在对磁盘进行 CRC 时，一般以扇区为基本处理单位。当磁盘 I/O 系统在进行写入扇区操作时，会自动计算要写入该扇区内容对应的 CRC 校验码。在将

数据写入扇区完毕后，接着将计算出得到的 CRC 校验码写到该扇区 CRC 参数记录位置，作为以后读该扇区成功与否的判断依据。在后续每次读取该扇区的数据时，磁盘 I/O 系统都会计算所读到的扇区其数据对应的 CRC 值，并与扇区保存的对应 CRC 值进行比较，如果相符，则说明读出的扇区数据正确，否则该扇区 CRC 值比对未通过，磁盘 I/O 系统会采用固执的重读策略对该扇区进行再次读取。此重读过程反复多次，十分耗费时间，这会导致计算机系统的运行速度严重变慢。

4.5.4 磁盘适配器

如前所述，磁盘子系统的组成可划分为接口—磁盘控制器—磁盘驱动器—盘片。在不同的磁盘子系统中，控制器与驱动器之间的逻辑划分可能不同；从装配级看，它们所处的位置也可能不同，相应地，各部分之间采用的接口标准也就不同。

在多数微型计算机系统中采取这样的逻辑划分：将磁盘控制器与接口合为一块磁盘适配器，一端插入主机系统总线的插座槽口中，另一端用电缆与磁盘驱动器相连接。驱动器在组装方式上可以采用内置式或外接式，小容量磁盘驱动器常内置于主机机箱之内，大容量磁盘驱动器则往往作为一个单独的外围设备，置于主机机箱之外，通过电缆连接。

1. 磁盘适配器简述

在以上逻辑划分中，磁盘适配器的逻辑组成和功能大致包括：① 与主机系统总线的接口逻辑，如端口地址选择，读/写数据、命令、状态信息的输入/输出、中断请求逻辑、DMA 请求逻辑等；② 磁道格式控制；③ 数据缓冲存储器；④ 串并转换，写入时将数据总线来的并行数据（字节或字）转换为串行写入数据序列，读出时将磁道的串行数据组装成字节，并行送往数据总线；⑤ 读/写控制，写入时将串行数据编码为 M2F 制或其他记录方式的记录码序列，读出时将驱动器送来的读出信号序列分离出同步信号与数据信号序列（为了使读时钟信号与读出信号频率一致，通常用锁相电路实现频率自动调整）；⑥ 按照某种接口标准（如 ST506）与驱动器连接，向驱动器发出命令，接收驱动器状态信息，收发读/写信号序列。现在，磁盘适配器中常用微处理器或单片机作为控制核心，相应地将控制程序固化于 ROM 之中，称为智能控制器。有关磁盘适配器的逻辑组成已在 5.5.5 节中介绍。

2. 磁盘驱动器逻辑结构举例

磁盘驱动器的逻辑组成与功能主要是：① 驱动盘片旋转；② 选择磁头，实现磁头的寻道定位；③ 进行读/写操作，写入电流的放大驱动，读出信号的放大；④ 按照某种接口标准（如 ST506）与适配器连接等。图 4-44 给出了一种磁盘驱动器的逻辑粗框，反映了软盘驱动器、小型硬盘驱动器的大致组成。

（1）控制电路

通常以单片机为核心，实现对驱动器的编程控制。常用的单片机如 MC6801、MC6803、Intel 8048 等，控制程序可以固化在 EPROM 中，如 2716（2 KB）。

（2）读/写信号序列接口

驱动器与适配器之间的读写数据传输采用串行方式，常用 RS-422 串行接口连接。

（3）接口逻辑

接收适配器发送来的有关命令，如驱动器选择、磁头选择、寻道方向选择、寻道步进脉冲、读写命令等。向适配器提供有关驱动器的状态信息，如回答驱动器被选中、准备就绪（主轴电动机达到额定转速、磁头已定位于 0 号磁道）、寻道完成、0 号磁道信号、索引脉冲、写故障等。

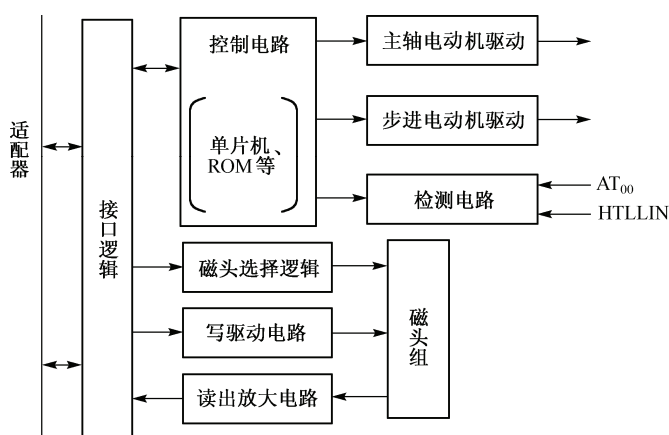


图 4-44 磁盘驱动器原理框图

(4) 写入驱动电路、读出放大电路、磁头选择及磁头组件

适配器发送来的写入信息，是按照某种记录方式进行编码的记录码序列，经过电流放大产生足够幅度的写入电流波形，送入选中的磁头线圈，使记录磁层产生完全的磁化翻转，即能由负向磁饱和翻转到正向磁饱和，或者由正向磁饱和到负向磁饱和。

由磁头读出的感应电势是很小的 (μV 级)，需经过放大电路放大，再经接口发送给适配器。读出放大器应尽量靠近磁头，否则弱信号在长距离传送中会受到干扰。

磁头选择靠译码器实现，根据适配器发送来的磁头选择信号，译码输出某个磁头的选取信号。某一时刻，驱动器中只有一个读/写磁头工作，单道、逐位地串行读/写。

磁头组件是一组读/写磁头，有关它们的一些工作状态信息发送给单片机，以供判别，如有无写故障等。

(5) 两种电动机驱动系统

主轴电动机一般采用直流电动机，靠一套伺服驱动电路驱动，作稳速旋转。为此，常需速度反馈调节。图 4-44 是采用步进电动机驱动的磁头定位系统。适配器根据寻道信息发送出寻道方向信号与步进脉冲，驱动器中有一套步进电动机的驱动电路。每接到一个步进脉冲，步进电动机使磁头按寻道方向步进一个磁道。

(6) 检测信号

为了提供 00 道检测与索引检测两种基准信号，一般采用霍尔传感器，将压力变换为电信号。

当磁头小车退到 00 道检测传感器附近时，发出 00 道检测信号 AT00。经控制电路处理，形成 00 磁道信号 TRKZERO (数字信号)，送往磁盘适配器。

如前所述，软盘片上有一索引孔。相应地，在驱动器中有光电检测电路，盘片每转一周，将产生一个索引检测信号。

硬盘驱动器中一般在主轴电动机附近装一个霍尔传感器。主轴每转一周，传感器产生一个索引检测信号 HALLIN。经控制电路处理，形成索引脉冲 INDEX (数字信号)，送往磁盘适配器。

3. 磁盘适配器的工业标准

目前在硬盘系统中，常将磁盘控制器与驱动器合成为一个设备整体，即硬盘存储器设备或称硬盘机。磁盘适配器只剩下接口功能，可以作为单独的多功能适配器插件插在主机箱中 (适配器的一端插入主机系统总线插座槽口，逻辑上与系统总线相连；另一端采用电缆与机外的硬盘存储器相连)，也可以直接做在主板上。当用户需要在自己的计算机系统中装入硬盘机时，必须注意

选择一种合适的硬盘接口方式才能适应自己的计算机系统。

现在常用的硬盘接口类型有 SCSI、IDE 和 SATA 接口。

(1) SCSI 接口

SCSI 接口即小型计算机接口 (Small Computer System Interface), 是在美国 Shugart 公司开发的 SASI (Shugart Associates System Interface) 的基础上, 增加了磁盘管理功能后组合而成的。其本质是采用 IBM 公司提出的 I/O 通道结构方式, 使一台智能外设能在单一总线上与多台主机进行通信。SCSI 接口方式当初是为磁盘设备设计的, 目前使用 SCSI 接口最多的也是磁盘, 但就其实质而言, SCSI 是一种用于适配器和智能控制器界面上的统一的 I/O 总线。SCSI 接口有很高的通用性, 除用于磁盘外, 还广泛用于光盘、CD-ROM、磁带、扫描仪、打印机等, 是一种多用途的 I/O 接口。其传输速率达 4 MBps, 属于高级智能型接口, 是 ANSI 标准化后公布的 X3T9.2 标准。

SCSI 可连接 8 台设备, 这些设备可以是磁盘驱动器, 也可以是其他外设或主机。这些设备可分成发送命令的启动设备和接收及执行命令的目标设备, 但这不是固定的, 各设备都可以作为目标设备或启动设备。SCSI 有几种连接方式: 单主机单控制器方式、单主机多控制器方式和多主机多控制器方式。在多主机多控制器的连接方式中, 每台设备均有识别号 ID, 每台设备还可带 7 台外设, 这些外设称为逻辑设备 LUN, 并分别给以编号。SCSI 可以在主机和外设之间、外设和外设之间, 以及主机之间进行通信, 且非常灵活。

(2) 磁盘 IDE 接口

前面介绍的硬盘接口对各种计算机都是通用的, 只要主机和磁盘驱动器之间的控制器分别符合它们之间的接口约定就可以了。IDE (Integrated Device Electronics) 接口是为 AT 微机及其兼容机专用的, 因此这种接口又称为 AT-BUS 或并行 ATA (Parallel ATA) 接口。

ATA (Advanced Technology Attachment) 接口标准从 ATA-1 一直发展到 ATA-7 (即 ATA133), 其支持最大仅为 133MB/s 的数据传输速度, 最远传输距离也仅为 0.5 米。

IDE 接口是一种早期的智能化驱动器电子接口, 它的主要特点如下:

① IDE 是在早期硬盘接口的基础上改进而成的, 其最大特点是把控制器集成到驱动器中, 在硬盘适配卡中不再有控制器这部分电路。

② IDE 实际上是系统级的接口, 除对 AT 总线上的信号作必要的控制外, 其余信号基本上是原封不动送往硬盘驱动器, 所以它采用命令线和控制线合用的 40 芯电缆线, 将接口适配器和 IDE 驱动器连接起来。这个接口适配器只是由一些地址译码和总线缓冲等少数电路组成, 有些设计者甚至将这个接口电路集成到系统主板上, 40 芯接口电缆直接插到主机上, 省出了一个扩展槽。

③ 采用高性能的旋转音圈电动机, 再配以嵌入式的扇区伺服机构, 定位精确, 速度快, 其传输速率达 7 MBps 以上, 且省电、抗震性能好, 断电后磁头能自动锁定在启停区。

④ 接口卡不需另配驱动程序, 所有 IDE 硬盘均由系统 BIOS 利用保存在 CMOS 中的硬盘参数直接驱动。对不同的 IDE 硬盘, 必须在 CMOS 中为其设置并保存相应的硬盘参数, 这组参数的设置直接影响系统对硬盘的驱动效果。

IDE 硬盘驱动器大多数是 3.5 英寸的微型驱动器, 在驱动器内部已装有控制器 ASIC 芯片。它已把通常的控制器和驱动器合成一体, 不但能完成寻道、读、写等驱动器功能, 而且可以完成将主机命令转换成控制器的操作, 即完成通常的控制器功能。

(3) 磁盘 SATA 接口

SATA 是 Serial ATA 的缩写, 即串行 ATA, 它是一种完全不同于并行 ATA 的新型硬盘接口类型, 由于采用串行方式传输数据而得名。

早在 2001 年，由 Intel、IBM、希捷等几大厂商组成的 Serial ATA 委员会就正式确立了 SATA 1.0 规范。到 2002 年的时候，又正式确立 SATA 2.0 规范，并于 2009 年正式发布了 SATA 3.0 规范。SATA 接口需要硬件芯片的支持，例如 Intel ICH5(R) 芯片，如果计算机主板南桥芯片不能直接支持的话，就需要选择第三方的芯片，如 Silicon Image 3112A 芯片等，这样就会引起一些接口硬件性能上的差异，且接口的驱动程序也会比较繁杂。

和传统的 IED (Parallel ATA) 接口相比，SATA 总线使用嵌入式时钟信号，具备了更强的纠错能力，与以往相比其最大的区别在于能对传输指令（不仅仅是数据）进行检查，如果发现错误会自动矫正，这在很大程度上提高了数据传输的可靠性。此外，SATA 采用的电源接口是 15 针（3 针为一组，共 5 组），数据接口也只有 7 针，这使得其在结构上比 IDE 接口更简单。SATA 还使用了差动信号系统，能有效地将噪声从正常信号中滤除。因为具备良好的噪声滤除能力，这使得 SATA 只需使用较低的驱动电压（0.5 V）即可。与 IDE 接口高达 5 V 的驱动电压相比，SATA 接口设备的功耗更低，其驱动 IC 的成本也更便宜。

使用普通的 IDE 接口磁盘，最高只能达到 133 MB/s 的数据传输速率，目前采用 SATA 3.0 接口标准的磁盘的数据传输速率最高可达 600 MB/s，大约是 IDE 接口的 4.5 倍，其理论传输距离也更远，甚至可达 2 米。除此之外，SATA 接口还支持热插拔。

4.5 光学存储及器件

光盘存储器的功能、部件和结构组成与磁盘存储器相似，是计算机系统的另一种重要大容量辅助存储器。

4.5.1 光存储原理

光存储技术是用激光照射存储介质，通过激光与介质的相互作用，使介质发生物理和化学特性上的变化，从而将信息存储下来的一种技术。光存储技术的基本物理原理可以概括为：当存储介质受到激光照射后，介质的某种性质（如反射率、反射光极化方向等）将发生改变，用表示介质特性的两种不同状态就可以与二进制代码 0 和 1 相对应，从而实现了对二进制数据的存储，而存储介质上数据的读出，则通过识别光学存储单元特性的变化来实现。

光盘就是利用光存储技术于近代发展起来的一种光学存储器件，其信息的写入与读出，均需要通过激光束处理信息记录介质才能完成，因而又被称激光光盘，简称光盘。如果按信息记录原理的不同，可分为形变型、相变型和磁光型等。如果从读写特性角度出发，还可以将它们划分为两种类型：只读型和读写型光盘。光盘的读写特性与信息记录原理之间也存在十分紧密的联系，下面从信息记录的特性角度分别进行介绍。

1. 存储介质的特性

(1) 形变型

存储介质经激光照射后，会在光盘记录薄膜上形成小孔（凹坑）或微小气泡，这种介质的物理形变一般是不可逆的，因此这种记录原理常用于只读型光盘。

光盘的基片材料一般采用聚甲基-丙烯酸甲酯（简称 PMMA），它是一种耐热的有机玻璃，基片厚度约为 1 mm 左右。记录介质为碲（Te）合金薄膜，厚度约为 0.035 μm 。

盘片结构有两种。一种被称为夹层结构，在两张基片上涂敷碲合金薄膜，然后将两张盘片粘接起来，记录面朝里，两张盘片之间形成一个空腔，其中充以惰性气体，使记录介质得到保护。

另一种被称为双层结构，在基片上先涂敷一层反射层（铝薄膜），反射层上涂一层二氧化硅保护膜，然后涂敷碲合金记录层，再涂一层透明的保护膜。也有人将它称为三层结构，即基片、反射层、记录层。

写入时，能量集中的激光光斑照射在某个区域，使该微小区域加热达到可熔点温度，保护膜与记录薄膜蒸发，留下一个凹坑（融坑），这就记录了一个 1。

读出时，低功率激光光束沿光道扫描。当照射到记录 1 的凹坑时，由于该处的记录层已蒸发掉，光束直接射到反射层，反射光强，经光检测器转换为读出信号。当照射到无凹坑处，由于记录层存在，反射光较弱，表示读出为 0。读出时激光功率只有写入时的 1/10，约 2 mW，不会使记录层发生新的形变。

这种形变一般是不可逆的，用于不可改写型光盘中。这种光盘用母盘复制而成，或由用户一次性写入。

（2）相变型

利用存储介质的晶相结构即结晶状态发生的可逆性变化，可以制成一种可改写型光盘。这种光盘盘片一般采取带沟槽结构，即在基片上预先刻制若干沟槽，在上面蒸发成一层记录介质薄膜，再涂一层树脂保护层。记录介质一般采用碲（Te）氧化物中渗入 5% 的锗（Ge）。沟槽宽度即光道宽度，典型值为 $0.5 \sim 1 \mu\text{m}$ 。槽深与所用激光的波长成正比，约为 $0.07 \mu\text{m}$ ，有槽处（即记录信息处）的记录层，厚约 $0.12 \mu\text{m}$ ；而无槽处记录层厚约 $0.05 \mu\text{m}$ 。

写入时，强激光照射 TeOx-Ge 记录薄膜，被照射的微小区域突然加热，Te 的粒子直径变大。由于照射时间极短，已变化了的粒子直径来不及再缩回去，就保持扩大了直径不变。粒子直径的变化引起元素结晶状态的变化，使光折射率增加约 20%，因此写入 1 的区域（照射过的区域）与写入 0 的区域（未照射的区域），在光的反射率上存在明显的差别。

读出时，弱激光对光道进行扫描，利用反射率的差异鉴别扫描位置记录信息是 0 还是 1。

擦除时，用激光照射记录介质，使记录信息区域退火，即加热后缓慢冷却。于是 Te 粒子的直径从大变小，恢复原有结晶状态，反射率差异随之消失，记录信息被抹去。

（3）磁光型

另一种可改写型光盘是磁光型光盘，它以磁性材料为记录介质，利用热磁效应写入信息，利用磁光效应读出信息，通过恢复原有磁化状态进行擦除。

这种光盘大多采用稀土类—铁族系非晶态磁性合金作为记录介质，如 TbFe 、 GdTbFe 等。盘片一般为三层结构：在 PMMA 基片上镀一层厚约 $0.1 \mu\text{m}$ 的记录薄膜如 TbFe ，其上涂一层氧化硅保护膜，然后镀一层银反射膜，再涂一层氧化硅保护膜。

写入前，记录介质在外加磁场作用下呈垂直磁化状态，称为自旋按垂直于薄膜的方向排列，如图 4-45(a)所示。

写入时，强激光照射的微小区域温度升高，自旋排列因热振动而被打乱，磁化强度下降。外加反向磁场使该区域的磁化方向发生翻转，即记录一位信息，如图 4-45(b)所示，这就是利用热磁效应记录信息。

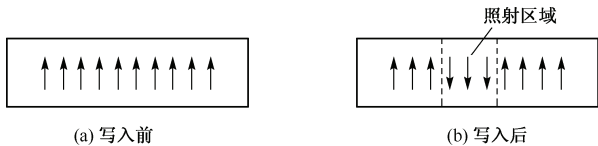


图 4-45 磁光型记录原理

读出时,弱激光照射记录介质,获得反射光。根据磁光效应(又称为克尔效应),反射光的振动方向与记录介质的磁化方向有关。当照射到向上磁化区时,反射光的偏振面将旋转一定角度;适当安置检偏器角度,使上述反射光不能通过检偏器。当照射到向下磁化区时反射光的偏振面将反向旋转同一角度,可以通过检偏器,经光电转换为读出信号 1。

擦除时,一方面用激光照射记录介质。一方面外加磁场,使介质回到“写入前”的状态,即清除了原存的信息。可见,在写入和擦除时都是利用热磁效应,即利用激光局部加热,改变加热区的磁特性,从而可用外加磁场改变磁化状态。

以上简要介绍了三种利用光来进行信息存储的工作原理。由于激光可以聚焦成极细的光斑,所以光盘的记录密度很高,约为同面积磁片的数十倍。目前,光盘早已成为了一种重要的大容量数据外存储器。

2. 激光波长与记录密度

从主要结构上看,普通 CD(Compact Disc)光盘、DVD(Digital Video Disc)光盘和 BD(Blue-ray Disc)光盘基本相同,只不过它们的厚度和存储介质材料各不相同。此外,这三种光盘的单片容量也存在显著差异,这主要与其相关的激光光束波长密切相关。一般而言,光盘片的记录密度受限于读出的光点大小,即光学的绕射极限(Diffraction Limit),其中包括激光波长 λ ,物镜的数值孔径 NA。所以,要提高光盘的记录密度,一般只能通过使用短波长激光或提高物镜的数值孔径使光点缩小。

对于 CD 光盘,其使用的激光(近红外光)其波长 $\lambda=780\text{ nm}$,物镜的数值孔径 $\text{NA}=0.45$,激光束会集到一点的距离需要 1.2 mm ,这就决定了 CD 光盘基板的厚度为 1.2 mm 。不管是 CD 光盘的基板过厚,还是过薄,激光束都不能会集到一点,从而严重影响数据的烧录和读取。

普通 DVD 光盘的激光(红光)其波长 $\lambda=650\text{ nm}$,物镜的数值孔径 $\text{NA}=0.6$,激光束会集到一点的距离只需要 0.6 mm ,这决定 DVD 光盘基板的厚度为 0.6 mm 。不过, 0.6 mm 的厚度太薄,制造出来的光盘也会因为太薄而容易折断。因此,在 DVD 的实际制造过程中会把两片 0.6 mm 厚的基板迭合在一起,共同组成 1.2 mm 的厚度。当然,在这种情况下,只有一片基板在记录数据,另一片基板则完全起保护的作用。

BD 光盘使用的激光(蓝光)其 $\lambda=405\text{ nm}$,物镜的数值孔径 $\text{NA}=0.85$,这就决定了蓝光产生的光点更小,从而使存储介质上的记录密度更大,也能做成单片容量更大的光盘。

4.5.2 光盘存储器

正如前文所述,光盘存储器是一种利用激光进行读写的大容量数据存储器,也是计算机的另一种重要辅助存储器,可以存放文字、声音、图像和动画等数字信息,属于非易失性存储器,其信息能永久保存,且体较小、价格低廉,普及广泛。

1. 光盘的基本特性

从分层物理结构上看,光盘一般至少要包括基板、信息记录层、光反射层这三层。基板是各功能结构层的载体,通常用聚碳酸酯材料做作,冲击韧性极好、使用温度范围大、尺寸稳定性好且无毒。记录层是真正用来刻录、保存数据的载体,一般可以用有机染料、碲合金薄膜或者稀土类磁性材料制作。反射层是用来反射激光光束的区域,借反射的激光光束读取光盘片中的资料,其材料一般为高纯度的纯银金属。光盘除了这三层基本结构外,根据具体的应用需要,还可增加保护层或者印刷层等。

从光盘的外观上看,120 型光盘普及最广,其外径尺寸为 120 mm,中心孔径为 15 mm,盘片厚度 1.2 mm,重约 14~18 g。其他外形规格的光盘,如 80 型等,用量稀少。目前,普通 CD 光盘的最大容量约为单面 700 MB,普通 DVD 光盘的单面容量约为 4.7 GB,BD 光盘则容量更大,其单面单层的数据总存储容量可高达 25 GB。

从信息分布形态上看,光盘与磁盘不同。磁盘是划分为若干分布均匀的同心圆磁道,而光盘的盘片一般则是由一条由内向外的螺旋线(类似蚊香)光道组成,光道的密度以及光道上的数据密度都远远大于磁道,如普通 CD 光盘,其相邻螺旋光道之间的距离约 1.6 μm ,道密度约为 16000 TPI。而对于 DVD 光盘,光道间距则已缩小至 0.74 μm ,BD 光盘则更小约为 0.32 μm 。光盘的光道又划分为若干等容量扇区。例如,容量为 650 MB 的普通 CD 光盘(数据模式)的光道由 333000 个扇区组成,每个扇区中可记录 2352 B 的数据,其中 304 B 为系统保留区,剩余 2048 字节才作为用户数据区。DVD 光道每个扇区(包括扇区头标 130 B)大小为 2697 B,其中真正用于用户存储数据的容量也仅为 2048 B。

对于光盘的数据编码,在逻辑上一般都采用“交叉交错理德-所罗门编码”(Cross Interleaved Read-Solomon Code, CIRC),其主旨是除了增加二维纠错编码外,还将源数据打散,再根据一定的规则进行扰频和交错编码,使数据相互交叉交错,这样能让用户数据的错误很难连续起来,有利于提高整体的纠错能力。CD 光盘数据的纠错一般由 4 B 的 ECC 码和 276 B 的 EDC 码(采用 CRC)来共同完成。DVD 与 CD 类似,但纠错的基本单位并不是一个扇区,而是一个纠错块(ECC Block, ECCB),一个纠错块包含 16 个物理扇区的数据区,共 32 KB,纠错块中对应每个扇区的纠错码则分散存储在这 16 个扇区中。

2. 光盘的分类

常见光盘从读写特性上可分为:只读型(Read-Only)光盘,如 CD-Video、CD-ROM、DVD-ROM 和 DVD-Video 等;读写型(Read-Write)光盘,如 CD-RW、DVD-RW 和 DVD-RAM 等。

从使用的光波来看,光盘主要分为:CD 光盘,使用近红外光,波长 780 nm,单张总容量一般为 650 MB;DVD 光盘,使用红光,波长 650 nm,单层总容量一般为 4.7 GB;BD 光盘,使用蓝光,波长 405 nm,单层总容量可达 25 GB。

总体上来看,光盘的发展趋势是面向大容量存储(如 DVD+R DL 产品),业界的技术研发也以此为导向。单面双层 DVD 盘片(DVD+R Double Layer)是利用激光聚焦的位置不同,在同一面上制作两层记录层。单面双层盘片在第一层及第二层的激光功率(Writing Power)相同(激光功率小于 30 mW),反射率(Reflectivity)也相同(反射率为 18%~30%),刻录时,可从第一层连续刻录到第二层,实现资料刻录不间断。

4.5.3 光盘驱动器及其发展方向

光盘是信息的存储载体,光盘驱动器则是实现对光盘进行信息写入和读出的一种光学驱动设备,故简称为光驱。

1. 光驱的工作原理

光驱器件一般由光驱主体支架、光盘托架和激光头组件等几部分构成,其中激光头组件最为重要,是光驱的核心部件。

① 激光头组件:包括光电管、聚焦透镜等部分,配合运行齿轮机构和导轨等,在通电状态下根据系统信号确定、读取光盘数据并通过数据带将数据传输到系统。

② 主轴驱动电机：提供光盘高速旋转所需的驱动力，在光盘的高速运行过程中提供快速、准确的数据定位功能。

③ 光盘托架：在开启和关闭状态下的光盘承载体。

④ 启动机构：控制光盘托架的进出和主轴马达的启动，加电运行时启动机构将使包括主轴马达和激光的头组件的伺服机构都处于半加载状态中。

⑤ 机芯电路板：控制伺服系统和控制系统的器件，是光驱的控制机构。

在光驱工作时，将数据先调制成二进制记录码，形成相应的写入脉冲序列，控制脉冲电流生成，再送入半导体激光器。激光器发出的激光，聚焦成能量高度集中的、只有微米数量级大小的光点，照射记录介质使其发生种变化，其工作原理如 4-46 所示。

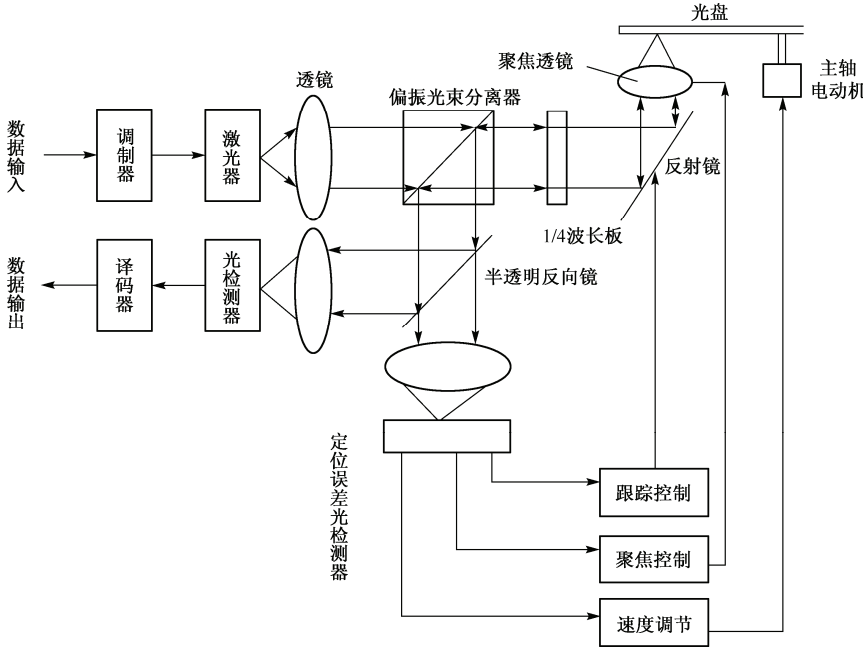


图 4-46 光驱的工作原理图

如图 4-46 所示，半导体激光器根据输入电流，以一定的功率发出光脉冲。激光光束经过透镜汇聚为平行光，通过偏振光束分离器、1/4 波长板，再经反射镜折向聚焦透镜，聚焦为小于 $1\text{ }\mu\text{m}$ 的细微强光点（又称光斑，红光为 $0.4\text{ }\mu\text{m}$ ，蓝光为 $0.14\text{ }\mu\text{m}$ ），照射于光盘记录介质上。写入数据 1 时，有脉冲信号输入激光器，最终产生细微光斑。写入 0 时，无脉冲信号输入激光器，因而无光斑。换句话说，激光束受写入数据代码的调制，转换为光斑的有无。直流电动机驱动光盘片旋转，光脉冲将在光道上形成一串有间隔的记录点。光头由定位系统驱动，可径向寻道，定位系统则用音圈电动机为执行电动机。

读出时，激光器加较低的直流电压，输出一小功率的连续激光束。其功率约为写入时的 10%，因此不会破坏光盘上已经写入的信息。扫描光盘时，激光器发出的激光仍然经过透镜、偏振光束分离器、1/4 波长板、反射镜、聚焦透镜，聚焦为直径小于 $1\text{ }\mu\text{m}$ 的光斑后照射光盘记录介质层，再通过光盘的反射层形成反射光。反射光经透镜、反射镜折向 1/4 波长板、偏振光束分离器，和入射光束相分离，折向半透明反射镜，经由光电转换器将光信号转变为电信号，即读出信号，经过反向调制后译码成相应的数据代码。

对于普通 CD 光盘而言，图 4-33 中调制器一般是按 EFM（Eight-Fourteen Modulation，8 比特

输入, 14 比特输出, 另加入 3 比特附加码, 记为 8-17) 方式进行调制, DVD 则是按 EFM+ (8 比特输入, 16 比特输出, 无附加码, 记为 8/16) 方式进行调制, 蓝光采用的是 EFMCC 调制, 其平均码率低于 CD 的 8-17 和 DVD 的 8-16, 接近于 8-15。

2. 光驱的分类

光驱是计算机里比较常见的一个配件。随着多媒体的应用越来越广泛, 光驱在台式机诸多配件中的已经成标准配置, 常见的有 CD-ROM 光驱、DVD 光驱 (DVD-ROM)、康宝 (COMBO) 和刻录光驱动等。

CD-ROM 光驱又称为致密盘只读存储器, 是一种只读的光存储介质, 利用原本用于音频 CD 的 CD-DA (Digital Audio) 格式发展起来。

DVD 光驱是一种可以读取 DVD 碟片的光驱, 除兼容 DVD-ROM、DVD-VIDEO、DVD-R、CD-ROM 等常见的格式外, 一般还支持 CD-R/RW、CD-I、VIDEO-CD、CD-G 等光盘格式。

COMBO 光驱是一种集合了 CD 刻录、CD-ROM 和 DVD-ROM 为一体的多功能光存储产品, 蓝光 COMBO 光驱指的是既能读取蓝光光盘也能刻录 DVD 光盘的光驱。

蓝光光驱是一种能读取蓝光光盘的光驱, 一般向下兼容 DVD、VCD、CD 等格式。

可刻录光驱包括 CD-R、CD-RW 和 DVD 刻录机及蓝光刻录机等。其中, DVD 刻录机又分 DVD+R、DVD-R、DVD+RW、DVD-RW (W 代表可反复擦写) 和 DVD-RAM。刻录光驱的外观与普通光驱差不多, 只是其前置面板上通常都清楚地标志着写入、复写和读取三种速度。

3. 主要的性能指标

光驱性能指标参数是生产厂商产品推出过程中的标称值, 包括接口类型、数据传输率、平均寻道时间、纠错能力、稳定性、内部数据缓冲、多种光碟格式支持等。

(1) 读盘模式

① CLV (Constant-Linear-Velocity): 恒定线速度读取方式, 在低于 12 倍速的光驱中使用的技术, 为了保持数据传输率不变, 随时改变光盘的旋转速度。读写光盘内沿光道数据时, 盘片旋转速度要比读写外沿光道要快许多。

② CAV (Constant-Angular-Velocity): 恒定角速度读取方式, 用同样的速度来读取光盘上的数据。但光盘上的内沿光道数据比外沿光道数据传输速度要低, 越往外越能体现光驱的速度, 即最高数据传输率。

③ PCAV (Partial-CAV, 区域恒定角速度读取方式, 是融合了 CLV 和 CAV 的一种新技术, 在读取外沿数据时采用 CLV 技术, 在读取内沿数据时采用 CAV 技术, 提高了光驱的整体数据传输速度。

(2) 读盘速度

光驱的平均寻道时间比磁盘长很多, 早期的 CD 光驱高达 400 ms, 目前主流的 DVD 光驱虽已大幅降低了平均寻道时间, 但仍需要 75~95ms, 磁盘为 10ms 左右。这是因为光盘的光道更多, 定位精度要求更高, 且光头本身也比磁头复杂, 驱动速度较低。

光驱的读盘时的数据传输率通常用“倍速”来表示, 这个数值是指光驱在读取盘片最外圈时的最快速度。光驱的实际工作速度与读写模式以及光盘的种类密切相关。

对于 CD 光驱, 其 1 倍速相当于 150 KBps, 对于 DVD 光驱, 1 倍速相当于 1358 KBps, BD 光驱则高达 36Mbps。最快的 CD 光驱一般为 52 倍速, 数据传输率 7800 KB/s, DVD 光驱已发展到 24 倍速, 数据传输速率为 32592 KB/s, BD 光驱目前已发展到 15 倍速。

(3) 数据缓存

CD/DVD 典型的缓冲器大小一般只有 128 KB, 针对光驱具体型号的不同互有差异, 如外置式 DVD 光驱就采用了较大容量的缓存。

可刻录 CD 或 DVD 驱动器一般具有 2~4 MB 以上的大容量缓冲器, 目前有采用 8 MB 容量的高端光驱。

大容量的缓存主要用于防止缓存欠载 (buffer underrun) 错误, 同时可以使刻录工作平稳、恒定的写入。一般来说, 驱动器速度越快, 就需要更多的缓冲存储器, 以适应更高的传输速率。

受制造成本的限制, 光驱的缓存不可能制作到非常大, 但适量、足够的缓存容量还是选择光驱的关键指标之一。

4. 光驱的接口

早期的 CD 光驱其接口有两种: IDE 接口和 SCSI 接口。

普通计算机中一般采用 IDE 接口, SCSI 接口的光驱一般用于网络服务器。目前, DVD 光驱多采用 ATAPI/EIDE 接口或 SATA 接口。对于外置光驱, 其接口多采用 USB 接口方式。BD 光驱现在多采用 SATA 接口。外置的 BD 光驱则采用 USB 接口的居多。

5. 光驱的未来发展

自从 1991 年, 全球 1500 家 IT 厂商加入软件出版商协会中的 Multimedia PC Working Group, 公布了第一代多媒体个人计算机规格, 它带动了光盘的流行。第一张光盘的容量仅为 640 MB, 光驱的数据传输率为 150 KB/s (单倍速), 平均搜寻时间为 1 s。此后, 第二代以提高速度为目标的光驱逐渐发展, 从单倍速发展到超过 32 倍速。

第二代光驱的应用环境是光驱逐渐普及, 光驱支持的格式也渐渐多了起来。相对于用户不断发展的速度需求, 其速度慢的弱点也凸显, 因此提高速度成为各家制造厂商技术竞争的首要目标。

第三代光驱解决了速度问题, 此时速度已不是各厂商发展技术的主要目标, 各厂家纷纷推出新技术, 使光驱读盘更稳定, 发热量更低, 工作起来更安静, 寿命更长。伴随第三代光驱技术的逐渐成熟, 国内厂商也发展起来, 其产品也成为了市场主流。

经过几十年的发展, 光驱技术已经趋于成熟。早期的 CD 光驱现已退出历史舞台, DVD 和 BD 光驱正成为市场的主流。纵观各光驱厂商, 其产品在技术上略有不同, 但产品品质都臻于完善。未来的光驱可能会追求更强的纠错率, 更快的传输速度, 更稳定工作过程、更安静工作噪声、更低的功耗, 以及体积更小、更易于携带。

4.6 存储系统性能的改进措施

目前, 计算机系统的整体性能在很大程度上受制于其存储器子系统。对于存储器子系统中的各类存储器, 根据其不同特性采取适当的管理和技术措施, 可以提高计算机系统的整体性能。因此, 本节将从存储系统组织的角度, 讨论一些有关技术, 如提高信息吞吐率的并行主存系统 (适应于并行处理机和多处理机系统的存储系统)、提高 CPU 访存速度的高速缓存, 扩大用户编程逻辑空间的虚拟存储技术、按内容寻址的联想存储器等。

如果在一个计算机系统中只配置有一个主 CPU, 该系统一般就可以称为单机系统。在单机系统中提高存储器系统性能的常用措施有: 双端口存储器、并行主存系统、高速缓存, 以及虚拟存储器技术等。

4.6.1 高速缓冲存储器

在存储系统的概述中，我们简要介绍了高速缓存 Cache 的基本概念。Cache 位于主存储器与 CPU 的通用寄存器组之间，新型的 CPU 芯片常在芯片内集成 1~2 个 Cache，第一级 Cache 一般容量只有几 KB 到几百 KB，第二级 Cache 一般有几 MB，一些高端 CPU 甚至集成了第三级 Cache。Cache 用来存放当前最活跃的程序和数据，作为主存某些局部区域数据的副本，如存放现行指令地址附近的程序，以及当前要访问的数据区内容。

由于编程时指令地址的分布基本上连续，对循环程序段的执行往往要重复若干遍，在一个较短的时间间隔内，对存储器的访问大部分将集中在一个局部区域中，这种现象被称为程序的局部性。我们将这一局部区域的内容从主存复制到 Cache 中，使 CPU 高速地从 Cache 中读取程序与数据，其速率可比主存高 5~10 倍，这一过程由硬件实现。编程地址仍是主存地址，因此对程序员来说看到的仍是访问主存，因而 Cache 对用户是透明的。随着程序的执行，Cache 内容也会按一定的规则进行相应替换更新。

为了实现 Cache 的上述功能，需要解决这样一些问题：首先是 Cache 的内容与主存之间的映射关系；其次是如何实现地址转换，将访问主存的地址转换成对应的 Cache 地址；再次是对 Cache 的读出与写入方式；最后是更新 Cache 内容的替换算法。

1. 地址映射方式

主存和 Cache 之间的数据交换，都是以固定大小（如 512 B）的数据块为基本单位整体进行的，因此主存与 Cache 的存储空间都应被分别划分成若干个大小相同的数据存储块。假定某计算机的主存和 Cache 都按字节编址，其主存容量为 1 MB（地址码长度为 20 位），按每块 512 B 划分，共被划分成 2048 块，块序号为 0~2047；Cache 容量为 8 KB，每块也是 512 B，故 Cache 划分成 16 块，块序号为 0~15。

下面将以此为例介绍 3 种基本的主存-缓存地址映射方式。

(1) 直接映射

基本特征：主存分组而 Cache 不分组，主存中每组包含的数据块的数量与 Cache 中划分的数据块数量一样。

直接映射就是当主存-Cache 进行数据块交换时，主存中的每个数据块只能映射到与该数据块具有相同组内块序号的缓存数据块对应位置，如图 4-47 所示。

直接映射具有下列对应关系：

$$K = J \bmod 2^C$$

其中， K 为 Cache 数据块的序号， J 为主存数据块的全局序号， C 为用来表示 Cache 块序号的二进制代码位数， 2^C 实际上就是主存每组内包含的数据块数量。

在上例中，参数 $C=4$ 。

在图 4-47(a)中，主存共 2048 块，Cache 共 16 块，主存再被划分 128 组，每组 16 块（与 Cache 的块数相同）。按直接映射规则，主存的第 J （全局序号）个数据块只能映射到与其组内序号（ J 除以 16 的余数，即 $J \bmod 2^C$ ）相同的第 K 块 Cache 存储位置。

因此，在直接映射方式下，每个主存数据块只能复制到某个固定的 Cache 块。基本映射规律是：将主存的 2048 块按顺序分为 128 组，每组的 16 个数据块分别与 Cache 的 16 个数据块位置是一一对应的。具体而言，主存第 0 块、第 16 块、第 32 块、……、第 2032 块，共 128 块，这些块的全局序号 J 与 16 相除以后得余数 $K=0$ ，故它们只能映射到 Cache 的第 0 块对应的位置上。

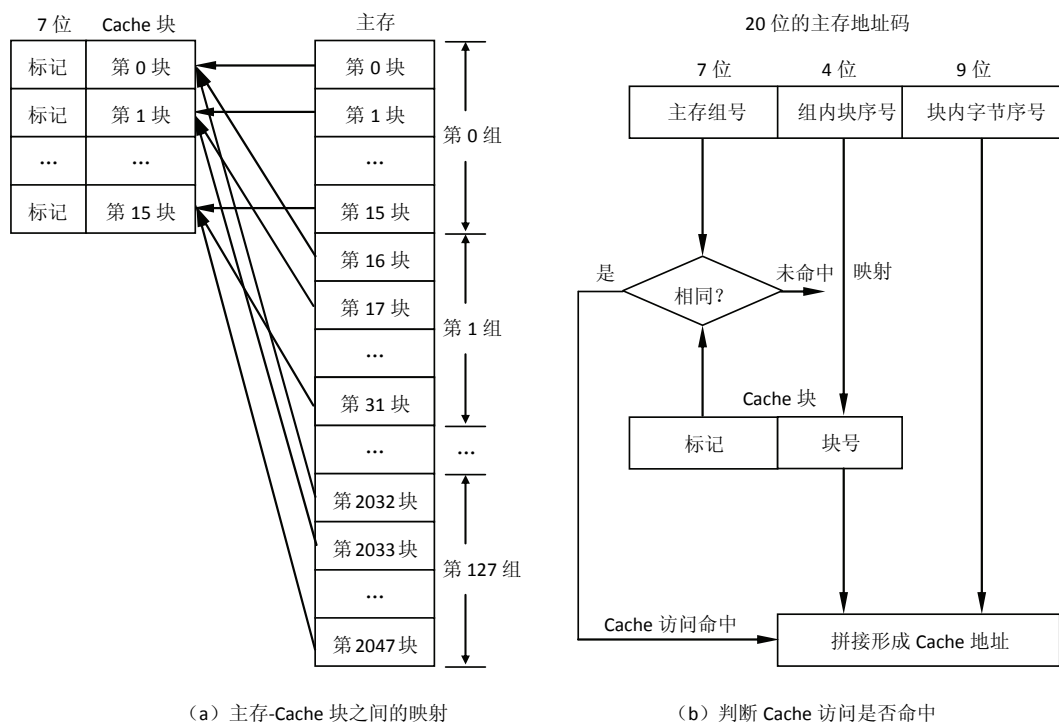


图 4-47 直接映射方式

同理，主存的第 1 块、第 17 块、第 33 块、……、第 2033 块，共 128 块，这些块的全局序号 J 与 16 相除以后得余数 $K=1$ ，故它们就只能映射到 Cache 第 1 块对应的位置上。以此类推，其余主存数据块在 Cache 中的映射位置，如主存第 15 块、第 31 块、……、第 2047 块，共 128 块，也只能映射到 Cache 的第 15 块对应位置。

当访问主存的数据时，CPU 会先给出一个 20 位的主存地址码，其中地址码的最高 7 位可以看成是主存分组后的组序号（范围：0~127），随后的 4 位可以看成是组内的块序号（范围：0~15），最后的 9 位则可以看成主存数据块中的字节序号（也称块内地址，范围：0~511），因此该 20 位的主存地址码在逻辑上就被分解成：组号（7 位）+组内的块序号（4 位）+块内的字节序号（9 位），其结构如图 4-47(b)所示。

如前所述，在具体映射时，主存的每组中都有一个数据块可以映射到 Cache 的同一块上，因而单靠主存地址分解得到的组内块序号，只能确定该主存块在 Cache 中可能的位置，并不能确定该主存块确实已被映射到对应的 Cache 块位置。例如，主存第 0 组的第 0 块、第 1 组的第 0 块、第 2 组的第 0 块都可以映射到 Cache 的第 0 块上，假设当根据主存的 20 位地址码访问主存第 1 组的第 0 块时，该块的直接映射位置为 Cache 的第 0 块，但 Cache 的第 0 块就一定是主存第 1 组的第 0 块映射过来的吗？答案是不确定的，因为也可能是主存第 0 组的第 0 块，或者第 3 组的第 0 块映射过来的。

为了准确判定 Cache 的某块具体是由哪一个主存块映射过来的，在 Cache 方面，为每块设立一个长度为 7 位的 Cache 组号标记，该标记恰与主存分组的组号对应。如果现在 Cache 的第 0 块是由主存第 16 块（全局序号）映射过来的，则其该 Cache 块对应的组号标记段为设置 1，用以标记当前的 Cache 块是由主存第 1 组中的某块映射过来的。根据直接映射规则，该主存块必须是第 1 组的第 0 块（组内序号）相对应。因此在访存时，只需两步就可以确定 Cache 访问是否命中：

第一步, 根据直接映射规则确定主存块对应的 Cache 块; 第二步, 比较主存地址码中的高 7 位 (主存组号) 与 Cache 块的 7 位组号标记, 如果二者相同, 则表明目标主存块已被映射到对应的 Cache 块中, Cache 访问命中, 否则访问未命中。

直接映射方式的优点是: 在硬件实现方面比较容易, 只需容量较小的可按地址访问的组号标志存储器和少量的比较电路, 硬件成本很低, 映射速度快。

其缺点是不够灵活、Cache 块的冲突概率很高, 可能使 Cache 的存储空间得不到充分利用。例如, 需将主存第 0 块和第 16 块同时映射到 Cache 中时, 由于它们都只能映射到 Cache 的第 0 块, 即使 Cache 的其他块空闲, 也始终会有一个主存块不能被映射到 Cache 中, 这会使 Cache 访问的命中率急剧下降。

(2) 全相联映射

基本特征: 主存和 Cache 均不进行分组。

全相联映射就是当主存-Cache 进行数据块交换时, 主存中的每个数据块可以随机映射到 Cache 中的任意一个块位置, 如图 4-48(a)所示。

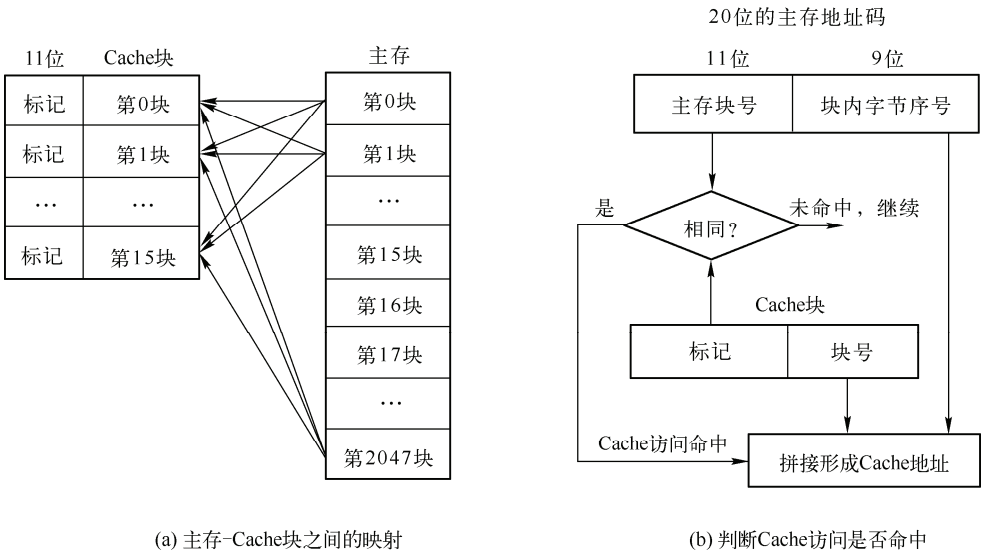


图 4-48 全相联映射的 Cache 组织

采用全相联映射方式时, 如果淘汰了 Cache 中某一数据块的内容, 则可调入任何一个主存块的内容, 因而比直接映射方式更加灵活, 但也存在一些严重缺陷。

在全相联映射方式中, 因为不存在数据块的分组, 则可以把 CPU 给出的 20 位主存地址码看成两部分, 把高 11 位可看成是主存的块号 (0~2047), 而把低 9 位看成是块内的字节序号 (0~511), 如图 4-48(b)所示。

由于每个 Cache 块可由 2048 个主存块中的任一块映射过来, 因此每个 Cache 块其标记字段也需要 11 位 (与主存块号对应), 这样才能通过它确定当前的 Cache 块是主存中的第几块映射得到的。因此, 在全相映射中, 与直接映射方式相比, Cache 块的标记字段的位数会增加, 这会导致其硬件比较逻辑的成本也增加。

采用全相联映射, Cache 块的冲突概率最低。只有当 Cache 块全部装满后才可能出现数据块冲突, 因此 Cache 的空间利用率最高。

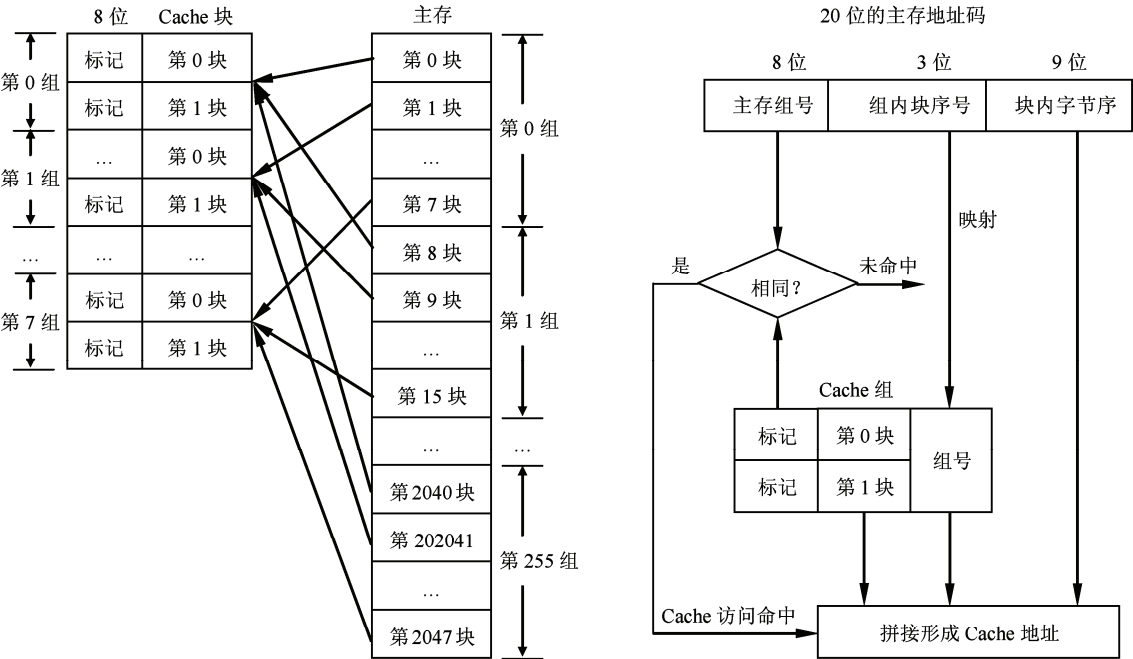
根据地址码访问主存时, 由于该地址码所在的主存块, 可以被映射到 Cache 的任何一块, 于

是要从 Cache 的第 0 块开始，把它的标记字段与该主存块的序号进行比较，相同则表示 Cache 访问命中，否则未命中。最好情况是第一次比较就判定 Cache 访问命中，最差情况是从 Cache 的第 0 块开始，逐一与全部 16 块的标记字段比较，直到找到符合的标记字段（Cache 访问命中），或者全部比较完后仍无符合的标记字段（Cache 访问未命中），最终才能判定本次 Cache 访问是否命中，因此全相联映射方式的速度比直接映射方式更慢，不能凸显缓存应有的高速性能。为了提高速度而把主存块号与各 Cache 块的标记字段并行比较，则比较电路会很复杂、硬件成本太高。

（3）组相联映射

基本特征：主存和 Cache 均要分组。先将 Cache 成若干组，每组若干个数据块，再将主存分组，主存每组包含的数据块数量与 Cache 划分的组数是一样的。

组相联映射就是指当主存与 Cache 之间以数据块为单位进行映射时，主存中的每一个数据块，只能映射到特定 Cache 组中的任意一个 Cache 块位置，此时要求主存块的组内块序号与 Cache 块所属的组号必须相等，如图 4-49(a)所示。



(a) 主存-Cache 块之间的映射 (b) 判断 Cache 访问是否命中
图 4-49 组相联映射的 Cache 组织

Cache 在分组时，若每组包括 2 个数据块，则 Cache 为 2 路组，此时的映射方式相应地被称为 2 路组相联映射，如图 4-49(a)所示。同理，若 Cache 每组包括 4 个数据块，则相应的映射方式称为 4 路组相联映射，其余情况以此类推。

在实际情况下，通常根据速度和命中率要求来设计组相联映射的路数，且 Cache 的存在对程序员是透明的，主存与 Cache 之间的地址变换和数据块替换都采用硬件来实现。

在直接映射方式中，主存分组，主存每组内的各块只能映射到一个唯一的 Cache 块，两者之间存在固定的映射关系，且主存各组中均有一块映射到某一个特定的 Cache 块。在全相联映射方式中，主存和 Cache 均不分组，两者之间以块为单位自由映射，没有固定的对应关系。在组相联映射方式中，主存和缓存均分组，主存每组内的各块只能映射到一个唯一的 Cache 组，但与 Cache

组内的各块是自由映射的，没有固定的对应关系。如图 4-49(a)所示，组相联映射时，在确定主存块应该映射到哪一个 Cache 组时，采用的是直接映射关系，当主存块映射到某个 Cache 组后，具体再映射到此 Cache 组内的哪一个 Cache 块，则采用的是全相连映射方式。由此可见，组相联映射实际上是直接映射和全相连映射的一种折中。

如前所述，组相联映射要求主存和 Cache 都分组，主存中各组内的块数与 Cache 分组的组数相同。如图 4-49(a)所示的 2 路组相联映射，Cache 划分成 8 组（范围：0~7），每组 2 块（范围：0~1），则主存被划分成 256 组（范围：0~255），每组 8 块（范围：0~7）。

两者之间的具体映射情况如下：

主存的第 0 块，即第 0 组的第 0 块，应映射到 Cache 第 0 组（两块中任意一个块的位置）。

主存的第 1 块，即第 0 组的第 1 块，应映射到 Cache 第 1 组（两块中任意一个块的位置）。

.....

主存的第 7 块，即第 0 组的第 7 块，应映射到 Cache 第 7 组（两块中任意一个块的位置）。

主存的其余块，以此类推。

试想若 Cache 分成 16 组，每组只包括 1 块（1 路组相联），此时主存应被划分位 128 组，每组 16 块。按组相联映射规则执行时，主存各组内的块先映射到对应的 Cache 组，但由 Cache 组只包括 1 块，此时主存块就只能映射到该块位置，故此时的 1 路组相联映射便退化成了直接映射。反之，若 Cache 只分成 1 组（组号为 0），每该组包含 16 块（16 路组相联映射，相当于 Cache 未分组），此时主存对应划分为 2048 组，每组也仅 1 块（组内的块号为 0）。按组相联映射规则执行时，主存块应先映射到组号也为 0 的 Cache 组中，由于这个唯一的 Cache 组包含有 16 个块，因此主存块可以进一步映射到这 16 个块中的任意块位置，可见此时的 16 路组相联映射等效于典型的全相连映射。

当需要访问主存时，CPU 会给出一个长度为 20 位的主存地址码，如图 4-49(b)所示。此地址码可以看成三段：高 8 位看成主存的组号（范围：0~255），随后的 3 位可以看成是组内的块序号（范围：0~7），最低的 9 位仍然看成主存块内的字节序号（范围：0~511）。因此，给定的任意主存地址码，都可以分解成三部分：**8 位组号+3 位组内块序号+9 位块内字节序号**。

由于采用组相联映射时，主存块是按其组内序号映射到特定的 Cache 组中的，进入 Cache 组后，由于每个 Cache 组中可能又有若干块，因此主存块再按全相联方式随机映射到 Cache 组内的某个块。换个角度看，在一个 Cache 组中，某个块也可能来自于主存的任意一个分组。对于某个 Cache 块，通过其所属的 Cache 组号可以断定该块是由主存分组中的第几块映射而来，但却不能断定数据块对应的主存组号。因此，为了确保与主存块所属组号的对应关系，需要为 Cache 的各块再增设一个 8 位的标记字段。

如图 4-49(b)所示，在判断 Cache 访问是否命中时，按 2 路组相联映射规，先将主存地址码中的第二部分（即组内的块序号，3 位）映射成 Cache 的组号（0~7），从而确定该主存块对应的 Cache 组。再将主存地址码中第一部分（即组号，8 位）分别与 Cache 组内的各块（序号 0 和 1）设置的标记字段（8 位）进行逐一比较。如果在该 Cache 分组中找到了与主存块所属组号一致的 Cache 块标记，则表明当前 Cache 访问命中，随即生成 Cache 地址码，据此把对主存单元的访问转化为对 Cache 单元的访问。如果比较结束，在 Cache 组内的两块中，均未找到与主存组号相同的标记字段，则表明 Cache 访问未命中。对于 2 路组相联映射，比较标记字段的次数最多为 2 次。

【例 4-12】某计算机按字节编址，Cache 共包括 16 块，若采用 2 路组相联映射，每块的大小

为 64 字节，主存第 268 号单元应映射到 Cache 的第几组？

Cache 分成 8 组，每组 2 块，则主存对应分成若干组，每组也应是 8 块。

对主存第 268 号单元， $268 \div 64 = 4$ 余 12，则该单元是主存第 4 块中的第 12 字节，且 $4 \div 8 = 0$ 余 4，则该单元属于主存第 0 组内的第 4 块。

组相联映射，主存块的组内序号与 Cache 的组号对应，则第 268 号单元应装入到第 4 组中。

由于 Cache 分组时，每组有若干块可以供主存块自由选择，因此它在完成主存与 Cache 之间地址映射方面比直接映射方式更灵活，命中率更高。Cache 每组内的页面数量有限，因而对标记字段进行比较所需付出的代价也不是很大，2 路组相联最多只需比较 2 次就可判断是否命中，而全相联映射中最多时需要比较 16 次才能判断是否命中，显然组相联映射比全相联映射速度更快。

2. 常用的替换算法

Cache 刚从主存调入数据块内容时，访问命中率较高。随着程序的执行，访问频繁地区将逐渐迁移，Cache 中的内容逐渐变得陈旧，这会使 Cache 的访问命中率下降，这就需要不断地对 Cache 内容进行更新。常用的替换算法大致有以下两类。

(1) 先进先出算法 (First In First Out, FIFO)

FIFO 的基本思想是：按调入 Cache 的先后顺序决定淘汰的顺序，在需要更新 Cache 块时，总是淘汰最先调入 Cache 的主存块。这种方法容易实现，系统开销（为实现替换算法而要求系统做的事所花费的时间和代价等）较小，但有些内容虽然调入较早，但可能仍在继续使用，因此这种替换算法也存在缺陷。

(2) 近期最少使用算法 (Least Recently Used, LRU)

LRU 的基本思想是：先为 Cache 的各块建立一个 LRU 目录，并按某种方法记录这些块的使用情况。当需要替换时，选择在最近一段时间内最久没有被使用或者使用频率最低的 Cache 块予以替换。显然，近期最少使用和近期最久未被使用这两种算法都属于 LRU 法，这是按调用频繁程度和使用情况决定淘汰顺序的，比较合理，能够使 Cache 的访问命中率较高，因而使用较多。但 LRU 比 FIFO 算法复杂，系统开销稍大。

3. Cache 的读写过程

(1) 读操作

访问主存时，一方面将主存地址送往主存，启动读主存，同时将主存地址送往 Cache 机构。按系统定义的映射方式从主存地址中提取或经过转换得到目标 Cache 地址，如主存组号、块号和块内地址，定位到 Cache 块并读取内容，并将相应的 Cache 块标记与主存地址中相应的地址标记进行比较，如果二者相同，则 Cache 命中访问，直接将 Cache 中的数据读出并送往访存源（如 CPU 中的某寄存器），放弃对主存内容的访问。

如果标记不符合，或者按映射方式搜索完毕全部 Cache 块后，仍未找到标记相符的 Cache 块，这就表明本次 Cache 访问未命中。此时只能从主存中读取数据并送给访存源，并且把该数据所在的主存数据块整体调入到 Cache 中，同时要修改 Cache 块相应的标记字段。

(2) 写操作

在程序执行过程中，常需将信息的处理结果写入主存，通常有以下两种写入方法。

一种方式是“回写”法 (Write-Back)。先只修改主存单元对应的 Cache 单元，并用标志予以注明，直到该 Cache 块从 Cache 中被替换出去时，才修改其对应的主存块。这种方式不需要在快速写入 Cache 时插入慢速的写主存操作，可以保持程序运行的快速性，但在写回主存之前，主存

块的内容与对应 Cache 块的内容可能不一致，有可能导致程序错误。

另一种方式称为“通写”法 (write-through)，即每次写入修改 Cache 时，同时写入修改 Cache 单元对应的主存单元。这种方式下，能使主存与 Cache 的内容始终保持一致，写入方式也比较简单，但需要插入慢速的主存访问操作，而且有些写修改过程有可能不必要，如暂存中间结果，因此这种方式的效率不高。

当 Cache 写未命中时，只能直接写入主存，此时是否将修改过的主存块调入到 Cache，有两种选择：一是将主存块立即调入到 Cache 中，称为 WTWA (Write Through with Write Allocate)，二是不将主存块立即调入到 Cache 中，称为 WTNWA (Write Through with No Write Allocate)。前一种方法保持了 Cache 与主存的一致性，但操作复杂。后一种方法操作简单，但 Cache 命中率会降低，主存块只有在读未命中时才调入到 Cache 中。

通写法是写 Cache 与写主存同时进行，优点是 Cache 每行不需设置一个修改位及相应的判断逻辑，缺点是 Cache 对 CPU 向主存的写操作无高速缓冲功能，这与 Cache 的设计初衷不太一致。

还有一种方式叫“一次写”法 (write once)。一次写法是一种基于“回写”又结合了“通写”的策略，即写命中和写未命中的处理与回写法基本相同，仅第一次写命中时需同时写入主存。这种策略主要用于某些处理器的片内 Cache，如 Pentium 处理器的片内数据 Cache 就采用的是一次写法。若对片内 Cache 块的再次或多次写命中，则按回写法处理。

4.6.2 虚拟存储器

4.1.2 节中曾简要介绍了虚拟存储器的基本概念，提到了虚存、实存、虚拟空间、实存空间、虚拟地址或逻辑地址、实地址或物理地址这样一些术语。

在采用了主存-辅存这样的层次结构来组织存储系统后，存储容量扩大了许多，就可以在存储管理部件（硬件）和操作系统中的存储管理软件的支持下，为用户提供一种虚拟存储器 (Virtual Memory, VM)。其总体容量大大超过 CPU 可以直接访问的主存，因而用户可在这个大空间里自由编程，不受主存容量限制，也不必考虑所编的程序将来在主存中的实际位置。虚拟存储技术对用户来说，自然是极有价值的，已在计算机中广泛使用。

采用虚拟存储器技术后，可将主存和辅存的地址空间统一编址，用户按其程序需要使用逻辑地址（即虚地址）进行编程。所编程序和数据在操作系统管理下先送入辅存（一般是磁盘），然后操作系统自动将当前急需运行的部分调入主存，供 CPU 处理，其余暂不运行部分留在辅存中。随程序执行的需要，操作系统自动按一定替换算法进行调换，将暂不运行部分由主存调往辅存，将新的存储内容由辅存调入到主存。

CPU 执行程序时，按照程序提供的虚地址访问主存。因此，先由存储管理硬件判断该地址内容是否已在主存中。若已调入主存，则通过地址变换机制，将程序中的虚地址转换为主存的实地址（即物理地址），据此访问主存的实际单元。若尚未调入主存，则通过缺页中断程序，以页为单位进行调入，以实现辅存与主存之间的内容更换。

上述过程对于用户程序是透明的，用户看到的只是用位数较长的虚地址编程，CPU 可按虚地址访存，可访问存储空间很大，可遍及辅存空间，显然这是一个虚拟层次。

操作系统编制者则需考虑主存与辅存的空间如何分区管理，虚实之间如何映射，虚实地址如何转换，主存与辅存之间如何进行内容调换等。其策略与 Cache 所用策略非常相似，一般有页式、段式、段页式三种方式。在高档微处理器中，已将有关的存储管理硬件集成在 CPU 芯片中，可以支持操作系统选用上述三种方式之一。

1. 页式虚拟存储器

将虚存空间与主存空间都划分为若干大小相同的页，虚存的页称为虚页，主存（即实存）的页称为实页。每页大小是固定，常见的有 512 B、1 KB、2 KB、4 KB 等。这种划分是面向存储器物理结构的，有利于主存与辅存间的调度。用户编程时也将程序的逻辑空间分为若干页，即占用若干虚页。相应的虚地址可分为两部分：高位段是虚页号，低位段是页内地址。

在主存中建立一种页表，提供为虚实地址变换的基本依据，并登记一些有关页面的控制信息。如果计算机采用多道程序工作方式，可为每个用户作业建立一个页表，硬件中设置一个页表基址寄存器，存放当前运行程序的页表的起始地址。

表 4-5 给出了一种页表组织示例，每一行记录了与某个虚页对应的若干信息。盘页号（块号）是该页在磁盘中的起始地址，表明该虚页在磁盘中的位置。控制位有若干位，例如：装入位（有效位）为 1，表示该虚页已调入主存，为 0 表示该虚页不在主存中；修改位指出对应的主存页是否被修改过；替换控制位为 1，表示对应的主存页需要替换；读写保护位指明该页的读写允许权限，如只允许读出（不能写入），或允许读出、允许写等。页表中必不可少的是实页号。如果该虚页在主存中，该项登记对应的主存页号。

表 4-5 页表组织示例

| 虚页号 | 盘页（块）号 | 控 制 位 | 实 页 号 |
|-----|--------|-------|-------|
| 0 | | | |
| 1 | | | |
| ... | ... | ... | ... |

图 4-50 表明了访问页式虚拟存储器时，虚实地址的转换过程。当 CPU 根据虚地址访问辅存时，首先将虚页号与页表起始地址合成，形成访问页表对应行的地址，根据页表内容判断该虚页是否在主存中。若已调入主存，从页表中读得对应的实页号，再将实页号与页内地址合成，得到对应的主存实地址。据此，可以访问实际的主存单元。

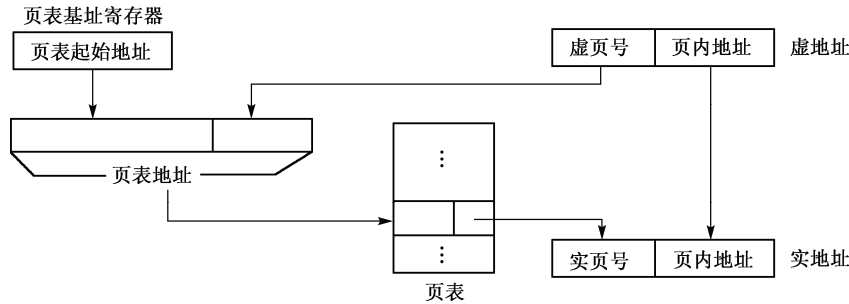


图 4-50 页式虚拟存储器地址转换示意图

若该虚页尚未调入主存，则系统产生缺页中断，以中断方式将所需的需页内容调入主存。如果主存空间已满，则需在中断处理程序中执行页面替换算法，将可替换的主存页内容回写入到辅存，再将所需虚页调入主存。

虚页面调进主存的方法可分为预调和请调两种。预调是指把不久即将用到的页面预先调进主存，在需要时就可立即在主存中访问，但要预测哪些页面将要用到是比较困难的。因此，较多使用的是请调方式，即发现当前 CPU 访问的虚页面不在主存时才产生缺页中断（或称调页中断），把虚页面调进主存。这种方法比较容易实现，但在需要访问主存时插入至少一页的调进，有可能影响系统响应速度。

页面调出的淘汰算法有先进先出算法 FIFO、近期最少使用算法 LRU 等，其算法思想与 Cache 替换算法相似。还有一种最优算法 OPT，即事先预测主存中各页将被访问的先后顺序，将最后才被访问的页面内容调出。这种算法虽然合理，但太理想化，不易实现，因为在程序运行以前要预测主存页面的访问顺序几乎是不可能的。

页表一般是保存在主存中的，故当 CPU 按虚地址访存时，首先要访问主存中的页表以进行虚实地址的转换，这就增加了访问主存的次数，降低了有效工作速度。为了将访问页表的时间降低到最低限度，许多计算机中除设置基本慢表外，还增设了快表(Translation Lookaside Buffer, TLB)。

在虚拟存储器中，普通的页表就是慢表，系统根据辅助存储器和主存的情况对其进行初始设置，慢表一般保存在主存中。快表一般是用一个专用的快速小容量存储器（一种按内容查找的联想存储器）构成，其容量一般很小，大约有 8~16 行，只能存放 8~16 个当前最常用虚页面对应的页表项，可按虚页号并行查询，能迅速找到对应的实页号。完全由专用硬件构成的快表，可以快速实现虚实地址转换，速度比访问主存中的页表快很多。

【例 4-13】 某计算机存储器按字节编址，虚拟（逻辑）地址空间大小为 16 MB，主存（物理）地址空间大小为 1 MB，页面大小为 4 KB。系统运行到某一时刻时，页表的部分内容和 TLB 的状态分别如表 4-6 所示，表中的页框号及标记字段为十六进制。

表 4-6 某时刻的页表和快表存储器状态

| 虚页号 | 有效位 | 页框号 | ... | 组号 | 有效位 | 标记 | 页框号 |
|-----|-----|-----|-----|----|-----|-----|-----|
| 0 | 1 | 06 | | 0 | 0 | - | - |
| 1 | 1 | 04 | | | 1 | 001 | 15 |
| 2 | 1 | 15 | | | 0 | - | - |
| 3 | 1 | 02 | | | 1 | 012 | 1F |
| 4 | 0 | - | | 1 | 1 | 013 | 2D |
| 5 | 1 | 2B | | | 0 | - | - |
| 6 | 0 | - | | | 1 | 008 | 7E |
| 7 | 1 | 32 | | | 0 | - | - |

请回答下列问题：

- (1) 虚拟地址共有几位，哪几位表示虚页号？物理地址共有几位，哪几位表示页框号（物理页号）？
- (2) 虚地址 001C60H 和 024BACH 所在的页面是否在主存中？若在主存中，则该虚拟地址对应的物理地址是什么？若不在，则请说明理由。

解答要点：

- (1) 由于虚拟地址空间大小为 16MB，且按字节编址， $16M=2^{24}$ ，所以虚拟地址长度应为 24 位。由于页面大小为 4KB， $4K=2^{12}$ ，则页内地址需 12 位，又有 $24-12=12$ ，故虚页号为虚地址的高 12 位。
- 由于主存（物理）地址空间大小为 1 MB，按字节编址， $1M=2^{20}$ ，所以物理地址共有 20 位，而页内地址 12 位，所以 $20-12=8$ ，即主存地址的高 8 位为页框号。
- (2) 虚地址 001C60H，则 C60H 为页内地址，剩余为虚页号即 001H=1，查表 4-6 可见虚页号为 1 的那行，即第 1 行的有效位为 1、对应页框号为 04H。
- 故 001C60H 所在的虚页面已在主存中，其对应的物理地址为页框号 04H 与页内地址 C60H 直接拼接，结果为 04C60H。

页表 \longleftrightarrow TLB 之间四路组相联，而 TLB 可存 8 个页表项，则每组存 4 个页表项，共分成 2

组。此外，页表也应按每组 2 行进行分组。

而虚页号是 12 位，且每个虚页在页表中对应一行，故页表中应有 212 行，按每组 2 行分组，共分成 211 组。所以页表分组后，虚页号可以被看成：组号 11 位+组内序号 1 位。

虚地址是 024BACH，则 12 位的虚页号为 024H（0000 0010 0100），而其高 11 位是组号，其低 1 位是组内序号，则分解成：组号 0000 0010 010（012H）+组内序号 0。

对组相联映射，页表项的组内序号=TLB 的组号，因此虚地址 024BACH 所在虚页所对应的页表项应映射到 TLB 的第 0 组中去。

因此在表 4-6 中分别对比组号为 0 的 4 个 TLB 行，结果显示第 0 组的最后一行中，标记字段为 012H，它与虚地址 024BACH 所在的虚页面对应的组号 012H 一致，且有效为 1、页框号为 1F，据此可确定虚地址 024BACH 所在的虚页面已在主存中。

虚地址 024BACH 对应的主存地址为页框号与页内地址直接拼接，即 1FBACH。

此外，如果计算机采用多道程序的工作方式，慢表可以有多个，但全机一般只有一个快表。增设了快表后，可以将页表中当前最常用的虚页面对应的页表项按一定的映射方式存放到快表中，快表存放的是慢表当前最活跃部分的副本。采用快、慢表结构后，快表项和慢表项之间的映射关系，同等于 Cache 块与主存块之间的关系，因此可以进行直接映射、全相联映射和组相联映射。系统访问页表时，先根据虚页号优先访问快表，若该虚页号存在于快表中（TLB 命中），则迅速将其映射成对应的实页号，并形成访问主存的实地址。若该虚页号不在快表中（TLB 未命中），则只能依靠访问慢表的结果来实现虚实地址转换，此时还要按某种预设规则考虑是否更新快表中的页表项内容。

2. 段式虚拟存储器

在段式虚拟存储器中，将用户程序按其逻辑结构（如模块划分）分为若干段，各段大小可变。相应地，虚拟存储器也随程序的需要动态地分段，并将段的起始地址与段的长度写入段表之中。编程时使用的虚地址分为两部分：高位是段号，低位是段内地址。如 80386 的段号 16 位，段内地址（又称为偏移量）32 位，可将整个虚拟空间最多分为 64 K 段，每段最大可达 4 GB，使用户有足够大的选择余地。

典型的段表结构如表 4-7 所示，其中装入位为 1，用来指示该段是否已经调入主存。如果该段已在主存中，则段起点登记其在主存中的起始地址。与分页方式不同，段的长度是可变的。段表还包括其他一些信息，如读、写和执行权限等。

表 4-7 段表结构

| 段号 | 装入位 | 段起点 | 段长 | 其他控制位 |
|-----|-----|-----|-----|-------|
| | | | | |
| ... | ... | ... | ... | ... |

段式虚拟存储器的虚实地址变换与页式地址变换相似，如图 4-51 所示。当 CPU 根据虚地址访存时，首先将段号与段表本身的起始地址合成，形成访问段表对应行的地址，根据段表内装入位判断该段是否调入主存。若已调入主存，从段表读出该段在主存中的起始地址，与段内地址（偏移量）相加，得到对应的主存实地址。

注意：在页式虚拟存储器中，页的大小固定，且为 2^n 字节（ n 为整数），所以页的划分是固定的，只要将实页号与页内地址两段拼装，即得到主存地址。而在段式虚拟存储器中，段的大小不固定，取决于程序模块的划分，因此在段表中给出的是段的起始地址，与段内偏移量相加，才能得到主存地址。

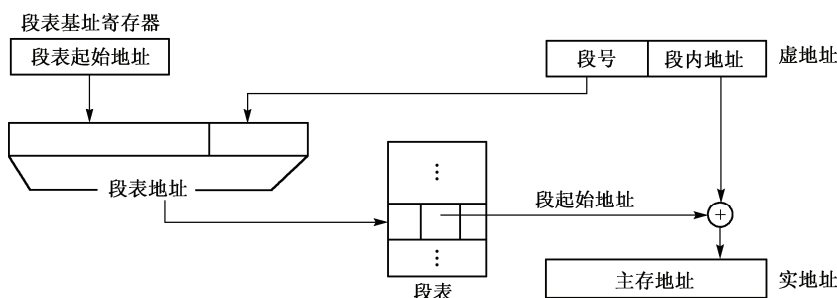


图 4-51 段式虚拟存储器地址转换

段式虚拟存储器的调进、调出及替换算法，与页式虚拟存储器相似，不再重复。

3. 段页式虚拟存储器

如前所述，页的大小固定，且都取 2 的整数幂个字节。所以在页式虚拟存储器中，可以将虚拟空间与实存空间进行静态的固定划分，与所运行的程序无关。假定某程序需占用两页半，则可填满前两页，仅第三页留有半页空区，称为零头，其他的程序需从新的页面开始。由于页表可按页表项提供虚实映射关系，所以一个程序所占的各页之间不必连续，如某程序占有实页号可以为 0、2、5。当一个程序运行完毕后，所释放的页面空间可以按页为单位地分配给其他程序。由此可见，页式虚拟存储器是面向存储器自身的物理结构分页的，有利于存储空间的调度和利用。但是，单靠页面的划分不能反映程序的逻辑结构，一个在逻辑上独立的程序模块本该作为一个整体处理，但可能被机械地按大小划分在不同页面里，这给程序的执行、保护和共享带来许多不便。

段式虚拟存储器则是按程序的逻辑结构进行段的划分，一个在逻辑上独立的程序模块可作为一段，可大可小。因此，存储空间的分段与程序的自然分段相对应，以段为单位进行调度、传送和定位，有利于对程序的编译处理、执行、共享和保护。但段的大小可变，不利于存储空间的管理和调度。一方面，段内必须连续，因各段的首、尾地址没有规律，地址计算比页式管理稍为复杂一些。另一方面，当一个段的程序执行完毕，新调入的程序段可能小于回收的段空间，各段之间会出现空闲区（即所谓零头，碎片），造成浪费。

为了综合分页和分段这两种方式的优点，许多计算机采用段页式虚拟存储器技术。在这种混合方式中，将程序先按其逻辑结构分成若干段，每段再分为若干大小相同的页面，主存空间也相应划分为同样大小的若干页。相应地，建立段表和页表，分两级查表实现虚实地址的转换。这样，系统既可以实现以页为单位的存储器管理，如调进或调出主存，也可以实现按段来共享和保护程序及数据。

如果计算机采取单道程序工作方式，则虚地址包含段号、段内页号和页内地址三部分。如果采用多道程序工作方式，则虚地址包含基号（用户标志号）、段号、段内页号和页内地址四部分。如图 4-52 所示，每道程序都有自己专用的段表，这些段表的起始地址存放在段表基址寄存器组中。相应地，虚地址中每道用户程序有自己的基号（又称为用户标志号），根据它选取相应的段表基址寄存器，从中获得自己的段表起始地址。将段表起始地址与虚地址中的段号合成，得到访问段表对应行的地址。从段表中取出该段的页表起始地址，与段内页号合成，形成访问页表对应行的地址。从页表中取出实页号，与页内地址拼装，形成访问主存单元的实地址。

段页式虚拟存储器兼有页式和段式的优点，但要经过两级查表，即先要查询段表再查询页表才能完成虚实地址的转换，因此耗费的时间更多，速度也更慢。

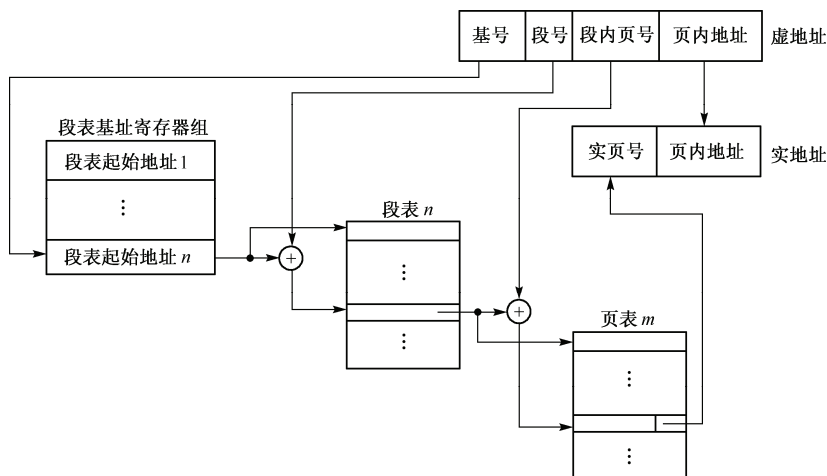


图 4-52 段页式混合虚拟存储器地址转换

4.6.3 双端口存储器

常规存储器是单端口存储器，即每次只能接收一个地址，访问一个编址单元，从中读取或存入一个字节或一个字。在执行双操作数指令时，就需要分两次读取操作数，工作速度较低。在高速计算机系统中，主存储器通常是信息交换的中心，一方面，CPU 频繁地访问主存，从中读取指令和存取数据；另一方面，外围设备也需较频繁地与主存交换信息。而单端口存储器每次只能接受一个访存者的读或写操作，这也影响了整体的工作速率。为了提高存储器的性能，在某些系统或部件中常使用双端口存储器，且已有集成的存储芯片可用。

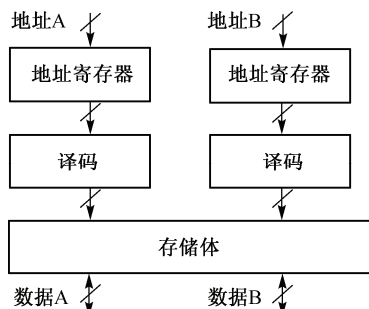


图 4-53 双端口存储器

图 4-53 所示双端口存储器具有两个彼此独立的读/写口，每个读/写口都有一套独立的地址寄存器和译码电路，可以并行地独立工作。两个读/写口可以按各自接收的地址，同时读出或写入，或一个写入而另一个读出数据。与两个独立的存储器不同，双端口存储器两个读写口的访存空间相同，可以访问同一区间甚至同一单元。

双端口存储器的常见应用场合如下。① 在运算器中采用双端口存储芯片，作为通用寄存器组，能快速提供双操作数，或快速实现寄存器间传送。② 让双端口存储器的一个读/写口面向 CPU，通过专门的存储总线（或称局部总线）连接 CPU 与主存，使 CPU 能快速访问主存；另一个读/写口则面向外围设备或输入/输出处理机 IOP，通过共享的系统总线连接，这种连接方式具有较大的信息吞吐量。此外，在多机系统中常采用双端口存储器甚至多端口存储器，作为各 CPU 的共享存储器，实现多 CPU 之间的通信。

4.6.4 并行存储器

如前所述，存储器系统速度的提高跟不上 CPU，这成为限制系统速度的一个瓶颈。因此，在高速的大型计算机中普遍采用并行主存系统，可在一个存取周期中并行存取多个字，从而依靠整体信息吞吐率的提高来解决 CPU 与主存之间的速度瓶颈问题，其中又分为单体多字方式和多体交叉存取方式两种。

1. 单体多字并行主存系统

如图 4-54 所示, 多个并行存储器共用一套地址寄存器, 按同一地址码并行地访问各自的对应单元, 如读出沿这 n 个存储器顺序排列的 n 个字, 每个字有 w 位。假定送入的地址码为 A , 则 n 个存储器同时访问各自的 A 号单元。我们也可以将这 n 个存储器视为一个大存储器, 每个编址对应于 $n \times w$ 位, 因而称为单体多字方式。

单体多字并行主存系统适用于向量运算一类的特定环境。在执行向量运算指令时, 一个向量型操作数包含 n 个标量操作数, 可按同一地址分别存放于 n 个并行主存之中。例如, 矩阵运算中的 $a_i b_j = a_0 b_0, a_0 b_1, \dots$, 就适于采用单体多字并行存取方式。

2. 多体交叉存取方式的并行主存系统

在大型计算机中使用更多的是多体交叉存储器, 如图 4-55 所示, 一般使用 n 个容量相同的存储器, 或称为 n 个存储分体。它们具有自己的地址寄存器、数据线、时序等外围电路, 可以独立编址地同时工作, 因而也称为多体单字交叉访问存储器。

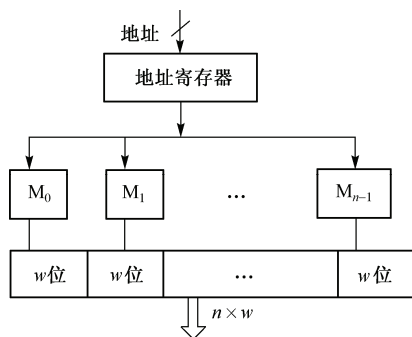


图 4-54 单体多字并行主存系统

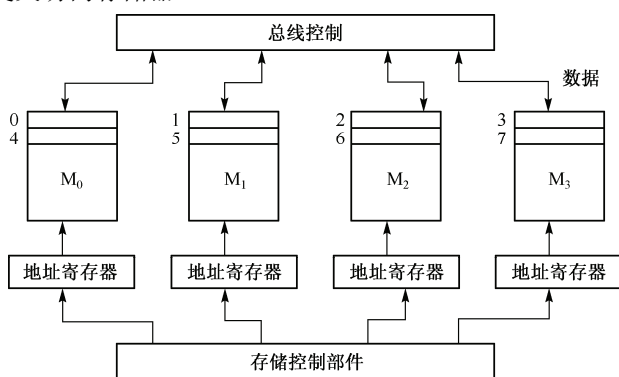


图 4-55 多体交叉存取并行主存系统

各存储分体的编址大多采用交叉编址方式, 即将一套统一的编址, 按序号交叉地分配给各存储体。以 4 个存储体组成的多体交叉存储器为例: M_0 体的地址编址序列是 0、4、8、12、..., M_1 是 1、5、9、13、..., M_2 是 2、6、10、14、..., M_3 是 3、7、11、15、...。换句话说, 一段连续的程序或数据将交叉地存放在几个存储分体中, 因此整个并行主存系统是以 n 为模交叉存取的。

相应地, 对这些存储体采取分时访问的时序, 如图 4-56 所示。仍以四存储体为例, 模等于 4, 各体分时启动读/写, 时间错过 $1/4$ 存取周期。启动 M_0 后, 经 $T_M/4$ 启动 M_1 , 在 $T_M/2$ 时启动 M_2 , 在 $3T_M/4$ 时启动 M_3 。各体读出的内容也将分时地送入 CPU 中的指令栈或数据栈, 每个存取周期将可访存 4 次。

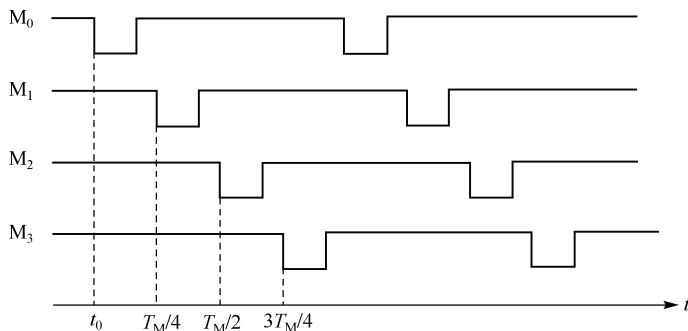


图 4-56 多存储体分时工作示意图

采取多体交叉存取方式，需要一套存储器控制逻辑，简称为存控部件。它由操作系统设置或控制台开关设置，确定主存的模式组合，如所取的模是多大；接收系统中各部件或设备的访存请求，按预定的优先顺序进行排队，响应其访存请求；分时接收各请求源发来的访存地址，转送至相应的存储分体；分时收发读写数据；产生各存储体所需的读/写时序；进行校验处理等。显然，多体交叉存取方式的控制逻辑比较复杂。

当 CPU 或其他设备发出访存请求时，控制部件按优先排队决定是否响应请求。响应后按交叉编址关系决定该地址是访问哪个存储体，然后查询该存储分体的“忙”触发器是否为 1。若为 1，表示该存储体正在进行读/写操作，则需等待；若该存储体已完成一次读/写，则将“忙”触发器置 0，然后可响应新的访存请求。当存储分体完成读/写操作时，将发出一个回答信号。

这种多体交叉存取方式很适合于支持流水线的处理方式，而流水处理方式已是 CPU 中一种典型技术。因此，多体交叉存储结构是高速大型计算机中的典型主存结构。

3. 并行处理机与多机系统中的存储组织

存储器的组织与计算机系统结构有密切的关系，它本身也是系统结构的一部分。前面提到的技术主要针对单机系统而言，而当前计算机系统结构的一个重要发展方向就是加强并行处理能力。传统的并行处理机一般指单指令流、多数据流结构，它有一个统一的指令部件和多个相同的执行部件（即处理单元）。各处理单元可以有自己的局部存储器或共享一组存储器。多机系统一般指多指令流、多数据流结构，各处理机可以独立执行自己的指令，因而可在任务级甚至指令级上实现并行处理，互连结构更为灵活、多样化。按各处理机间耦合的松紧程度，多机系统又可分为紧耦合和松耦合两大类。紧耦合多机系统通过共享存储器实现互连，松耦合多机系统则通过通信网络实现互连。

因此，在并行处理机与多机系统中，存储器按其在系统中的作用和地位，可分为局部存储器（本地存储器）和共享存储器两类。并行处理机，特别是多机系统，结构模式甚多，我们仅举几个例子，用以说明存储组织的典型模式。

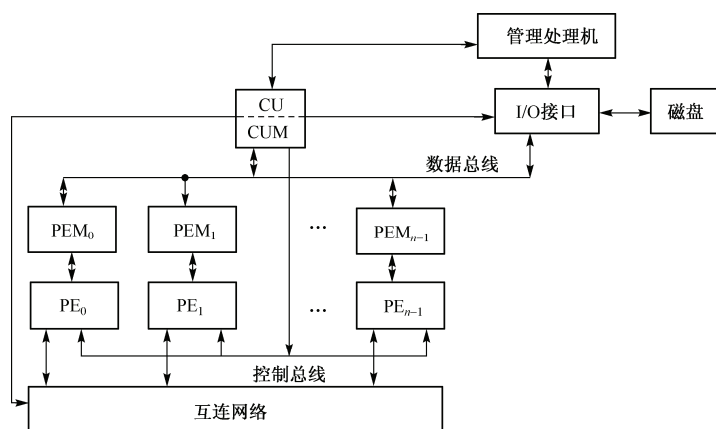
（1）并行处理机中的存储结构

在图 4-57 中，CU（Control Unit）是控制部件，PE（Processing Element）是一组相同的处理单元，即执行部件，PEM（Processing Element Memory）是各处理单元的本地存储器，即局部存储器。在图 4-57(a)结构中，将主存分为两部分：一部分集中在控制部件 CU 之中，用来存放系统程序 and 用户程序；另一部分是若干个可并行操作的局部存储器 PEM，分布在各处理单元 PE 中。因此，各个处理单元可以高速地从自己的局部存储器中获得数据。大容量后备存储器则由磁盘存储器承担。各处理单元 PE 可以通过互连网络实现交换，也可由其局部存储器 PEM 读出，送到控制部件 CU，再由 CU 经数据总线传送给另一个 PE。

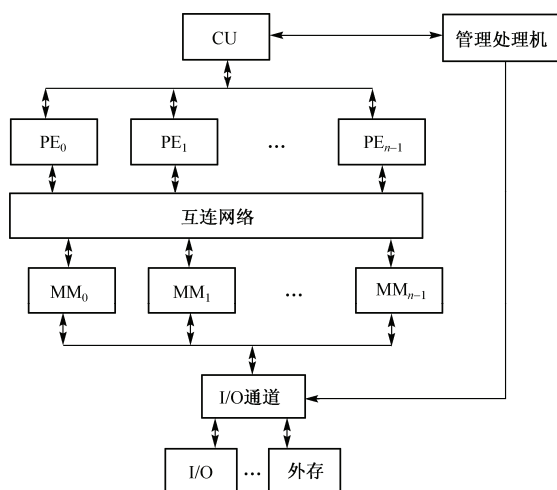
在图 4-57(b)所示结构中，将各处理单元的局部存储器集中在一起，构成一个统一的主存储器（Main Memory，MM），各处理单元 PE 可以通过互连网络共享这些主存储器，也可以实现高速并行访问。

（2）紧耦合多机系统中的存储结构

在小规模紧耦合多机系统中，常采用总线互连模式，如图 4-58 所示。各处理机有自己的本地存储器（Local Memory，LM，局部存储器），该 CPU 执行的程序和数据就存放在它的 LM 中。如果规模不大，可将 CPU 与其 LM 组装在一个插件上。总线上还挂接了共享存储器，各 CPU 可以通过公共总线访问它。共享存储器就像一个公用的信箱，通过它实现各 CPU 之间的信息交换。例如，CPU₀ 将信息写入共享存储器，而 CPU₁ 从共享存储器中读取该信息。



(a) 分布存储结构



(b) 共享存储结构

图 4-57 并行处理机的存储结构

一组总线的数据传输率有限，而且总线控制权的转换也要花费一些时间。因此，在高速系统或大规模系统中，常采用纵横开关实现多处理机与多存储体之间的共享型互连。如图 4-59 所示，每个 CPU 与一条水平通路连接，每个存储模块与一条垂直通路连接，行列交叉处有一连接部件，称为转接点。

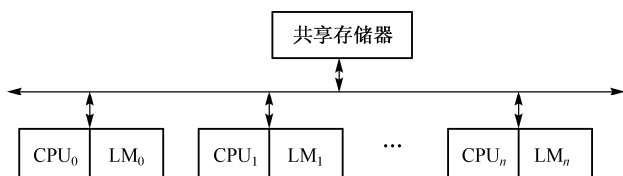


图 4-58 总线式多机系统的存储结构

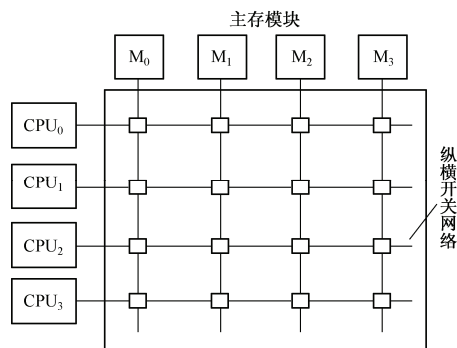


图 4-59 纵横开关式共享存储器

在任一时刻，各行、各列中只允许激活其中的一个转接点。如果是 4×4 纵横开关，则最多可连接 4 个处理机和 4 个存储模块，最多有 4 个转接点被激活，使 4 个 CPU 各访问一个存储模块，因而可同时传送 4 路数据。如果两个或两个以上的 CPU 同时请求访问同一个存储模块，则属于冲突，转接点将排队，轮流响应。

纵横开关内部的逻辑比较复杂，其复杂程度与成本不亚于 CPU，已有集成芯片供选用。在大规模并行处理系统中往往采用这样的结构：用纵横开关芯片将若干 CPU 与相应的局部存储器连接为一组（如 64 个 CPU 芯片），再将若干组连接成（分级连接）大规模的多机并行处理系统，这样的系统可以连接成千上万个 CPU 及其局部存储器。

（3）松耦合多机系统中的存储结构

图 4-60 是一种典型的松耦合多机系统，通过消息传输系统 MTS 连接多个计算机模块。每个模块是一个可以独立的基本系统，CPU 通过局部总线连接局部存储器 LM、局部 I/O 设备。CAS 是通道及仲裁开关，实现局部总线与上一级通信线路的连接，常带一个通信用的缓冲存储器。

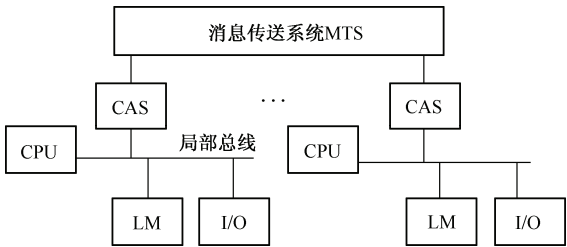


图 4-60 松耦合多机系统中的存储结构

消息传输系统 MTS 可以设计得很简单（如一级通信总路线），则 CAS 中的通信缓存为多个处理机模块所共享，以传递消息。MTS 本身也可以较复杂（如一个共享存储系统），由互连网络连接。MTS 也可以是一个多端口存储器，各处理机分占一个端口，从而实现通信。此外，MTS 也可以理解成是一种多级通信体系。

4.6.5 联想存储器

常规存储器是按地址访问的，即送入一个地址编码，选中相应的一个编址单元，然后进行读/写操作。而在信息检索一类工作中，需要的却是按信息内容选中相应单元，进行读/写。例如，从一份学生档案中查找某生的学习成绩，送出的检索依据是该生的姓名（字符串），找到相应的存储单元或存储区，从中读出他的成绩数据。当使用常规存储器进行检索时，就需要采用某种搜索算法，依次按地址选择某个存储单元，从中读出姓名信息（字符串），与检索依据进行符合比较。若不符合，按算法修改地址，再读出另一姓名信息，进行比较。直到二者相符，表示已找到所需寻找的学生姓名，然后按此找到对应的存储区域读出成绩数据。可见，用常规存储器进行信息检索，需将检索依据的内容设法转化为地址，因此效率往往很低。能否将有关的这些姓名与检索依据同时进行符合比较，一次就找到相符内容所存的单元呢？于是出现了“联想存储器”。

联想存储器（Associative Memory，AM），不是根据地址码寻址，而是根据所存储的信息的部分或者全部特征进行存取的，它是一种按内容寻址的特殊存储器。

为了实现输入信息（检索依据）与存储信息的符合比较，联想存储器的每位存储单元包含一位双稳态触发器、一位符合比较线路和读写电路。由于每位存储单元都有符合比较逻辑，所以输入信息可以同时与所有字进行对应位比较，称为字并行操作。而常规存储器是字串行操作，每次

只能读出一个字（一个编址单元）进行比较，一个字一个字地进行。所以使用联想存储器实现信息检索，可以一次找到相符合的单元，大大提高检索速度。显然，联想存储器的逻辑比常规存储器复杂得多，成本也就高得多，目前还只能生产小容量的集成电路芯片，如 256×48 b 的联想存储器芯片的存取周期为 100 ns。现在可达到的水平约是每片数十 Kb。因此，在实用中往往只将检索所需的关键信息存放在联想存储器中，如学号、姓名等，而其他数据（如成绩等）仍存放在较大容量的常规存储器中。

图 4-61 是一种以联想存储器为核心的信息检索系统硬件组成框图。检索中可能要查询的关键信息，即需要与之比较的信息，存放在可各自并行比较的联想存储器中， m 字 $\times n$ 位。这些信息是数据的特征位，如学号、姓名、科目号一类，总位数不超过 n 位。

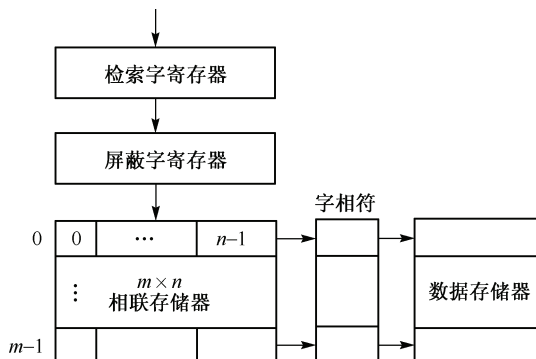


图 4-61 相联存储器工作原理图

输入的检索依据称为检索字，存放在检索字寄存器中。检索字寄存器的总位数与联想存储器的位数 n 相同，但每次检索时一般只用到其中的一部分，如只输入学号，或只输入姓名，检索字又称关键字或比较字。

因此设置一个 n 位的屏蔽寄存器，用来存放屏蔽字代码。如果本次检索只用到高 8 位信息，即输入的检索字为高 8 位，则屏蔽字的高 8 位为 1，而其余低位均为 0，将本次不用的无效位屏蔽掉。高 8 位有效检索信息将能与联想存储器中的 m 个字的高 8 位同时进行符合比较，其余被屏蔽掉的无效位将不进行符合比较。如果输入的检索字是另外的检索项，则修改屏蔽字。屏蔽字中为 1 的诸位就像一个窗口，只允许窗口对应的检索项进行比较操作。

如果联想存储器中某单元的对位内容内容与检索项完全符合，则将符合寄存器中的相应位设置为 1，表示该字就是要查找的字。符合寄存器有 m 位，每一位对应一个联想存储器的字单元。

若其他信息（如各科成绩）存放在另一个常规数据存储器中，则符合寄存器的各位经编码产生地址，据此到数据存储器中读/写。如果一个学生的各种成绩占用一段连续字节，则该地址就是对应区的首址，可连续读/写若干单元。

若全部信息（如果不是很长）都存放在联想存储器中，则符合寄存器中为 1 的位将直接指向对应的字单元，该位符合信息打开对应字单元的读/写控制门，可对其进行读/写。

在虚拟存储器中，需要建立段表和页表，如果这些表存放在联想存储器中，就能实现快速查找。在对大容量数据库、知识库进行检索的专用系统中，在进行逻辑推理的模式匹配中，以及多种联想型操作中，越来越多地使用联想存储器作为按内容寻址的支持。

习题 4

1. 简要解释下列名词术语

虚拟存储器 随机存储器 RAM 只读存储器 ROM 存取周期 数据传输率
动态刷新 直接映射 Cache 全相联映射 Cache 组相联映射 Cache
段页式虚拟存储器 联想存储器

2. 某半导体存储器容量 $8\text{ K}\times 8$ 位, 可选 RAM 芯片容量为 $2\text{ K}\times 4$ /片。地址总线 $A_{15}\sim A_0$ (低), 双向数据线 $D_7\sim D_0$ (低), 由 R/\overline{W} 线控制读写。请设计并画出该存储器逻辑图, 并注明地址分配与片选逻辑式及片选信号极性。

3. 某半导体存储器容量 $7\text{ K}\times 8$ 位。其中固化区 $4\text{ K}\times 8$, 可选 EPROM 芯片: $2\text{ K}\times 8$ /片。随机读写区容量为 $3\text{ K}\times 8$, 可选 SRAM 芯片: $2\text{ K}\times 4$ /片、 $1\text{ K}\times 4$ /片。地址总线 $A_{15}\sim A_0$ (低), 双向数据总线 $D_7\sim D_0$ (低), R/\overline{W} 控制读写。另有控制信号 \overline{MREQ} , 低电平时允许存储器工作。请设计并画出该存储器逻辑图, 并注明地址分配与片选逻辑式及片选信号极性。

4. 某机地址总线 16 位 $A_{15}\sim A_0$ (低), 访存空间 64 KB 。外围设备与主存统一编址, 即将外围设备接口中有关寄存器与主存单元统一编址, I/O 空间占用地址空间 $\text{FC00H}\sim\text{FFFFH}$ 。现用 2164 构成主存储器, 请设计并画出该存储器逻辑图, 并画出芯片地址线与总线的连接逻辑, 以及行选信号与列选信号的逻辑式, 使访问外设时不访问主存。

5. 某半导体存储器容量为 14 KB , 其中 $0000\text{H}\sim 1\text{FFFH}$ 为 ROM 区, $2000\text{H}\sim 37\text{FFFH}$ 为 RAM 区, 地址总线 $A_{15}\sim A_0$ (低), 双向数据总线 $D_7\sim D_0$ (低), 读写控制线 R/\overline{W} 。可选用的存储芯片有 EPROM (4 KB /片) 和 RAM ($2\text{ K}\times 4$ /片)。

(1) 计算所需各类芯片的数量。

(2) 说明加到各芯片的地址范围值和地址线。

(3) 写出各片选信号的逻辑式。

(4) 画出该存储芯片逻辑图, 包括地址总线、数据线、片选信号线 (低电平有效) 及读写信号线的连接。

6. SRAM 和 DRAM 分别依靠什么原理存储信息? 需要刷新吗?

7. 设某机主存 1 MB , 用 1 MB /片的 DRAM 芯片构成, 芯片最大刷新周期 2 ms , 问在 2 ms 之内至少应该安排几个刷新周期?

8. 某机主存容量 64 KB , 用 2164 DRAM 芯片构成。地址线 $A_{15}\sim A_0$ (低), 双向数据线 $D_7\sim D_0$ (低), R/\overline{W} 控制读写操作。请设计并画出该存储器逻辑图。对地址的行、列转换与数据线连接, 应有明确描述。

9. 某机主存容量 64 KB , 用 2164 DRAM 芯片构成。请为此设计一种动态刷新逻辑。

10. 动态刷新周期的安排方式有哪几种? 简述它们的安排方法, 并指出在下列情况中可选取哪一种或几种刷新方式。

(1) 教学用单板计算机。

(2) 常用个人计算机。

(3) 带多个分时终端的超小型计算机。

(4) 如果主存的存取周期 200 ns , CPU 平均访存时间约占主存工作时间的 90% 。

(5) 主存的存取周期 200 ns , CPU 平均访存时间约占主存工作时间的 40% 。

11. 若对磁表面存储器写入代码 10011, 请画出 NRZ1 制、PE 制、FM 制、 M^2F 制等记录方式的写电流波形。

12. 某磁盘有 100 个柱面, 每个柱面有 10 个磁道, 每个磁道有 128 个扇区, 每个扇区容量为 512 字节。该磁盘的存储容量是多少?

13. 某计算机字长为 32 位, CPU 主频为 500 MHz , 磁盘共有 16 个盘面, 512 个柱面, 每磁道包含 100 个扇区, 每个扇区 512 字节, 该磁盘旋转速度为 12000 RPM 。

- (1) 计算该磁盘总容量？
- (2) 计算该磁盘的数据传输率 (bit/s)？
14. 双端口存储器与两个独立存储器有何不同？
15. 何谓单体多字并行主存系统？何谓多体交叉存取并行主存系统？
16. 某机主存按字节编址，而系统总线数据通路宽 32 位。请提出一种连接方案，以粗框描述，并简要说明。
17. 如果要求半导体存储器在完成读/写后产生信号 **READY**，并通过系统总线通知其他设备。请设计该信号的产生逻辑。
18. 如果两台 CPU 通过各自的地址总线与数据总线共享一个半导体存储器，请为此设计存储器逻辑，使两组地址/数据线之间能够分离，且能处理访存冲突。
19. 如果需用常规存储芯片构成双端口存储器，允许存取周期延长。请设计此存储器的逻辑。
20. 试比较当前光盘与磁盘两者的记录密度、平均存取时间和数据传输率三项性能指标。
21. 主存和 Cache 之间的映射方式有几种？请简述每种方式的基本思想，并分析每种方式中如何通过变换内存地址以得到 Cache 的目标地址。
22. 何谓随机存取？何谓顺序存取？何谓直接存取？请各举一例进行说明。
23. 某计算机存储器系统参数如下：
 - ① TLB 共有 256 项，和页表之间按 2 路组相联方式组织映射。
 - ② 64 KB 的数据 Cache，块大小为 64 B，组织方式也是 2 路组相联。
 - ③ 虚拟地址 32 位，物理地址 24 位。
 - ④ 页面大小固定为 4 KB。

图 4-62 给出了系统的简单示意，请分别计算其中各字段 A、B、C、D、E、F、G、H 和 I 各段所占的位数，并给出计算过程。

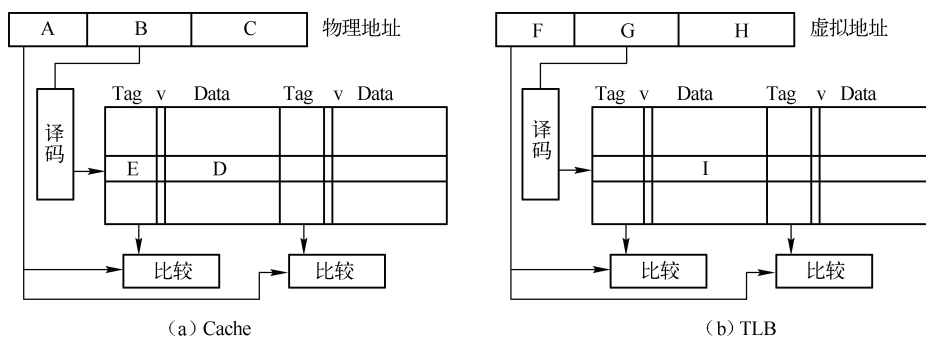


图 4-62 题 23 图

第5章 总线与输入/输出子系统

计算机硬件系统可以粗略地分成五大部分：运算器、控制器、存储器（内存）、输入设备（如键盘、鼠标）、输出设备（如显示器），这些部件必须连接起来，实现信息交互，并协调一致地工作，才能实现计算机的基本功能，即执行程序。在计算机中将连接这些部件的通信线路称为总线。

输入、输出设备是人们与计算机交互的平台，输入、输出设备通过 I/O 接口与总线连接，将数据输入处理器，把处理结果通过总线和 I/O 接口输出到输出设备。输入/输出子系统的硬件主要包括：输入、输出设备、输入/输出（I/O）接口和总线。

本章将在前面章节的基础上讨论计算机各组成部件的连接方式和信息交换手段，主要包含两方面的内容：部件之间的互连结构，各部件之间交换的数据和控制信号及管理互连结构所需的控制机制。连接结构的设计取决于部件之间所需交换信息的类型、交换信息的方式和信息的传输范围等因素。一般而言，所有的连接结构都要支持以下几类部件之间的传输。

- ◎ 存储器到 CPU 的传输：CPU 需要从存储器读取指令和数据。
- ◎ CPU 到存储器的传输：处理结果写入存储器。
- ◎ I/O 设备到 CPU 的传输：CPU 从 I/O 设备读取数据。
- ◎ CPU 到 I/O 设备的传输：CPU 向 I/O 设备发出数据或命令。
- ◎ I/O 与主存储器间的传输：I/O 设备通过硬件方式直接与主存储器交换数据，即直接存储器访问（DMA）方式。

在上述传输中，CPU 与主存储器之间的传输被视为主机内部的传输，除此之外的其他传输被视为主机系统与外部设备之间的输入/输出传输。

为了使结构规整、明了，便于管理和控制，目前的计算机系统大多以总线的方式来连接各大部件。由于外围设备的种类繁多，特性各异，通常外围设备都是通过一个转接电路连接在总线上的，这个转接电路即我们所说的接口。外围设备通过接口连接到总线，与主机系统通信，就可以使外围设备的设计独立于主机系统。

因此，在输入/输出子系统中涉及的内容包括：总线、接口、信息输入/输出的控制机构和相关的程序。信息输入/输出的控制机制不同，在接口的具体构成上也会有所不同。比如，以中断方式与主机交换信息的外部设备，在接口中就要设计能完成中断传输的有关部件和处理机制，这样的接口又常被称为中断接口。

“计算机组成原理”课程是从硬件的组成、设计和工作原理的角度来解释整个计算机硬件系统的，因此本章重点讨论的是输入/输出子系统中前三部分的内容。

5.1 概述

输入/输出子系统的主要功能是主机将外部数据通过某种控制方式传输到总线上，再传输到主存中保存，或者把主存中的数据通过总线并利用某种控制方式，将数据传输到指定的外部设备中去处理。从硬件的逻辑结构上看，输入/输出子系统的硬件组成至少应包括总线、接口、设备控制器及外部设备等。

接口的主要功能是连接计算机与外部设备。它们一般以插卡的形式插在计算机主板上的插槽中或直接集成在计算机主板上,其中一些公共接口逻辑(如中断控制逻辑、DMA 控制器和 USB 接口等)常配置于主板之上。早期的计算机系统(如 IBM PC/XT 机)中是用中小规模的集成电路(如 Intel 8259 中断控制器、8237 DMA 控制器、8250 串行通信接口等外设接口芯片)组成公共接口逻辑。随着集成电路技术的发展,芯片的集成度快速增长,设计了一系列的专用芯片把这些中小规模集成电路和并行接口芯片(如 Intel 8255)、定时电路(如 8253)等近百种接口芯片集成在一起,形成了现代计算机中常提到的芯片组来完成控制功能,但其控制的原理是类似的。所以,在中断及 DMA 的技术介绍中,涉及控制芯片时仍以单个的芯片功能进行介绍。

接口的一侧是面向外部设备的,这一侧应与外部设备在连接方式、数据格式及控制逻辑等方面保持一致,否则接口无法与外部设备交互;另一侧是面向符合某种标准的总线的,这一侧也应与总线在连接形式、数据格式及控制逻辑等方面保持一致,否则接口无法与总线交互。显然在接口的两侧连接特性不可能完全一致,因此在接口内部需要进行相应的转换,使总线能与外部设备进行信息交换。

总线上不仅连接有各种输入/输出接口,还连接有主存储器及 CPU,它们之间的信息交换都需要通过总线进行。总线通常一次只允许一个设备发送数据,但允许同时有一个或多个设备接收数据,如何确定总线控制权(拥有总线控制权的设备才能主导数据的收发)是总线设计时需要考虑的一个关键问题。

在软件层面,对外部设备的 I/O 操作会涉及三个层次的程序:

① 设备控制程序:固化在设备的控制器中,如磁盘控制器、打印机控制器等,其功能是控制外设的具体读、写操作,以及处理总线上的访问控制信号。

② 设备驱动程序:由操作系统、主板厂商或设备制造商提供的一组针对各种外围设备的驱动程序,为用户屏蔽了外围设备的物理细节,用户只需采用简单统一的操作界面来实现设备控制,如通过逻辑设备名调用外部设备。

③ 用户输入/输出程序:用户编写的对外部设备进行输入和输出操作的程序。

本节将对计算机部件的连接模式、总线和接口的通用功能和特性加以描述。后面将具体介绍几种主要的信息传输控制机制。

5.1.1 总线简介

总线是一种用来连接计算机各功能部件并承担部件之间信息传输任务的公共信息通道,能在各部件之间传输数据和控制命令。总线被多个部件分时共享,每一时刻只能有一个设备掌握总线进行数据收发,但多个设备可以同时从总线接收数据。

计算机中的总线一定是符合某种技术规范标准(即总线标准)的,在标准规范中定义了某一类总线机械结构规范、功能规范、电气规范及时间特性等方面应严格遵守的规定。只有约定了总线标准,才能使计算机中的各部件在逻辑上相互独立,部件的接口只要符合规定的总线标准,相互之间就能通过总线实现互连互通,从而使计算机硬件系统的层次结构更加清晰、模块化程度更高,计算机系统的扩展也更容易实现。

如前所述,计算机中需要通过总线进行传输的信息主要有三类,即数据信息、地址信息和控制信息,因此承载信息种类的不同,也有三类与之相适应的总线类型,分别是数据总线(Data Bus)、地址总线(Address Bus)和控制总线(Control Bus)。

从总线的层次结构方面来看,总线又有(芯)片内总线、局部总线、系统总线、I/O 总线、

通信总线的划分。比如，Intel 平台的前端总线属于局部总线。

按总线传输数据的不同格式，总线又有并行传输总线（Parallel Bus）和串行传输总线（Serial Bus）之分。比如，PCI 就是一种并行总线，而 RS-232 则是一种串行总线标准。

如果从总线数据传输所采用的控制方式来看，总线又分同步总线（Synchronous Bus）和异步总线（Asynchronous Bus）。顾名思义，同步总线就是利用同步时钟信号来控制数据的输入、输出，异步总线则是利用“主 - 从”设备两者的交互应答来控制数据的输入、输出。除这两种常见的总线控制模式外，还存在一种在标准的同步控制方式中，引入了延长时钟周期概念的总线控制方式，这种方式被看成标准同步控制方式的一种扩展，所以这类总线一般也被称为扩展同步总线。

通常，用总线宽度、总线频率和总线带宽等技术指标来评价一类总线的综合性能。在设计总线时，除了要遵守该总线的技术标准外，还应考虑设备如何申请总线，用什么方式对多个申请进行优先级仲裁，以及总线的控制方式及传输过程中的时序安排等因素。

5.1.2 接口的功能与类型

接口泛指设备部件（硬、软）之间的交接部分。主机（总线）与外围设备或其他外部系统之间的接口逻辑，称为输入/输出（I/O）接口，或称为外围设备接口。在现代计算机系统中，为了实现设备间的通信，不仅需要由硬件逻辑构成的接口部件，还需要相应的软件，从而形成了一个含义更广泛的概念，即接口技术。

软件模块之间的交接部分称为软件接口。例如，作为计算机与操作人员之间的接口操作系统中就有一个面向硬件、特别是外围设备的软件模块，叫做基本输入/输出系统 BIOS，固化在主机系统板的 ROM 中。BIOS 中有一个软件模块，叫做 BIOS 接口模块，用来连接各种不同版本的操作系统的其他软件模块，如连接命令处理程序与文件系统。

硬件与软件的相互作用，所涉及的硬件逻辑和软件又称为软硬件接口。例如，由硬件信号引发相应软件模块的调用，或者由软件执行产生的相应硬件信号。

1. I/O 接口的基本功能

I/O 接口一般位于总线与外围设备之间，负责控制和管理一个或几个外部设备，并负责这些设备与主机间的数据交换。总线通常是符合某种总线标准定义的，因而可以被符合这种标准的外围设备所通用，它不局限于特定的设备。外围设备各有其特殊性，如命令/状态含义、工作方式等都有可能不同，因此只能在它们之间设置接口部件，以解决数据缓冲、数据格式变换、通信控制、电平匹配等问题。一般而言，I/O 接口的基本功能可概括为以下几方面（作为一种具体的接口，其功能可能只有其中的一部分）。

（1）寻址

接口逻辑接收总线送来的寻址信息，经过译码，选择该接口中的某个有关寄存器。

（2）数据传输与缓冲

设置接口的基本目的是为设备之间提供数据传输通路，但各种设备的工作速率不同，特别是 CPU、主存与外围设备之间，往往差异较大。为此，需要在接口中设置一个或数个数据缓冲寄存器，甚至设置局部缓冲存储器，以提供数据缓冲和实现速度匹配。所需的缓存容量（如字节数）称为缓冲深度。

（3）数据格式变换、电平变换等预处理

接口与总线之间通常采用并行传输；接口与外围设备之间有可能采取并行传输，也有可能采

取串行传输，视具体的设备类型而定。因此，接口有可能需要担负数据的串并格式转换功能。

设备使用的电源与总线使用的电源有可能不同，因此它们之间的信号电平有可能不同。例如，主机使用+5 V 电源，某个外围设备采用+12 V 电源，则接口应实现信号电平的转换，使采用不同电源的设备之间能正常运行并进行信息传输。

更复杂的信号转换，如声、光、电之间的转换，一般由外围设备本身实现，不属于接口范畴。在采用大规模集成电路之后，一些专用型电子设备往往与接口做成一块插件，直接插入主机机箱或总线插槽，如语音 I/O 板、图像输入板等，这种情况下就没有必要机械地在物理意义上将设备与接口分开。

许多接口中采用微处理器、单片机（又称为微控制器）、局部存储器等芯片，可编程控制有关操作的处理功能大大超出了纯硬件的接口，这样的接口常称为智能接口。

（4）控制逻辑

主机通过总线向接口传输命令信息，接口予以解释，并产生相应的操作命令发送给设备。接口形成设备及接口本身的有关状态信息也通过总线回送给 CPU。

如果采用中断方式控制信息的传输，则接口中有相应的中断逻辑，如中断请求信号的产生、中断屏蔽、优先级排队、接收中断批准信号、产生相应的编码等，其中的部分逻辑也可能集中在公共接口逻辑中。

如果采用 DMA 方式控制信息传输，则接口中有相应的 DMA 逻辑，如 DMA 请求信号的产生、屏蔽、优先级排队、接收批准信号等。现在较多的逻辑结构是将 DMA 控制器与接口分开，由 DMA 控制器接收并保存 DMA 初始化信息（如传输方向、主存缓冲区首址、交换量），控制总线实现 DMA 传输。DMA 接口则接收外部设备寻址信息，从设备读出或向设备写入数据信息，通过总线送入主存或接收主存数据。

简单的接口根据总线时序信号或设备的时序信号工作；复杂的接口可能有自己的时序信号，如在串行接口中需要控制移位寄存器操作，实现数据的串并转换等。

2. I/O 接口的编址

在计算机系统中，设备的种类很多，即使是同一类设备，可能也会有多个。为了确保外设通过相应的设备接口能与主机正常通信和数据交互，就要求 I/O 接口能承担起主机和外设之间数据通信枢纽任务。I/O 接口中一般都设置有很多寄存器，如命令寄存器、地址寄存器、状态寄存器和数据缓冲寄存器等。在一次具体的 I/O 操作中，主机实际上只能利用 I/O 指令直接控制到 I/O 接口，再由设备控制器对外设实施具体的 I/O 控制。

外设对于主机而言，仅是个抽象存在的逻辑实体，主机对外设的寻址实际上只能通过对设备控制器中特定功能寄存器的寻址来间接实现。因此，在设计设备接口时必须考虑外设的编址方式。

① 外设单独编址。为设备接口中的每个寄存器（也叫 I/O 端口）都分配一个独立的端口编号（地址），这些端口的编号与主存单元的地址是无关的，它们不占用主存的单元地址。比如，系统为某个主存单元已分配总线地址码 0100H，此时仍可以为设备控制器中的某个寄存器分配端口地址 0100H，两者可以完全相同。

对于这种编址方式，需要设置专用的 I/O 指令，即“显式 I/O 指令”，而且在 I/O 指令中必须明确指明外设对应的 I/O 端口地址。显然，I/O 指令中指明的地址自然应该是设备控制器中的寄存器地址，也就不会被解析成主存单元地址了；同理，访存指令中给出的地址自然就是主存单元的地址，也就不会再被解析成接口寄存器的端口地址。

② 外设与主存统一编址。将一部分总线地址（低端地址）分配给主存使用，另一部分（高端地址）分配给设备接口中的寄存器（也叫 I/O 端口）作为寄存器端口地址。外设的这种编址方式中，接口寄存器的端口地址实际上占用了一部分本应分配给主存的地址。比如，某内存单元的总线地址为 1AFFH，则接口寄存器就不能再使用地址 1AFFH 了，两者不能相同。

对这种编址方式，不需设置专用 I/O 指令，因为接口寄存器已被处理成为特殊的主存单元，所以可以利用通用的访存指令来实现外设的 I/O 操作。此外，访存指令给出的地址虽然都是总线地址，但可以根据该地址的高低端分布情况来确定是对主存寻址还是对外设寻址，因此也不会发生混淆。能通过通用的访存指令来实现对外设的 I/O 操作，但其实质并未使用 I/O 指令，所以这类指令也常被称为“隐式 I/O 指令”。

3. I/O 接口分类

（1）按数据的传输格式划分

① 并行接口：接口与外部设备之间采用并行方式传输数据。

② 串行接口：接口与外部设备之间采用串行方式传输数据。

注意：接口与总线之间一般都采用并行方式传输数据，因此串行接口中一般需要移位寄存器，以及相应的产生移位脉冲的控制时序，以实现串并转换。

选用哪种接口，一方面取决于设备本身的工作方式是串行传输还是并行传输，另一方面与传输距离的远近有关。当设备本身是并行传输且传输距离较短时，一般采用并行接口。如果设备本身是串行传输，或者传输距离较远，需降低传输设备的硬件成本时，一般适合采用串行接口。例如，通过调制解调器（Modem）的远距离通信就需要串行接口。

（2）按时序控制方式划分

① 同步接口：与同步总线连接的接口，接口与总线间的信息传输由统一的时序信号控制，如 CPU 提供的时序信号或者专门的总线时序信号。接口与外部设备之间允许有独立的时序控制操作。

② 异步接口：与异步总线相连的接口，接口与总线间的信息传输采用异步应答的控制方式，无统一的时序控制信号。

（3）按信息传输的控制方式划分

① 直接程序传输方式接口：接口中设置有状态寄存器保存设备的当前状态，CPU 可以通过 I/O 指令查询接口中的状态字，然后进行相应的输入和输出控制操作。

② 中断接口：接口中有中断系统所需的逻辑，通过该接口主机与外围设备之间的信息传输采用程序中断方式进行。

③ DMA 接口：接口中有 DMA 逻辑，通过该接口主机与外围设备之间的信息传输采用 DMA 方式进行。

将接口设计成中断接口后，也可以不按中断方式来工作，只采用程序查询方式来实现信息的传输控制。换句话说，中断接口往往覆盖了程序查询方式接口的功能。事实上，一些接口（如磁盘存储器接口）既有 DMA 功能，也有中断功能，这样的接口既是 DMA 接口，也是中断接口。

此外，通道是比一般的 DMA 接口更复杂的控制器，甚至在通道的基础上发展出的 IOP（I/O Processor，输入/输出处理器）或者 PPU（Peripheral Processor Unit，外围处理机），但它们已超出了一般接口的层次概念。

上面这些分类方法是从不同的角度出发的，各种分类标准并不矛盾。因此，常常可以同时从多角度来描述一个接口的特性，如同步中断并行接口。

5.1.3 输入/输出控制方式

为了适应大数据、高性能计算服务的需求，现代计算机系统的硬件规模越来越大，体系架构也变得越来越复杂，因此对系统的 I/O 操作提出了越来越高的要求，这使得计算机系统采用的输入/输出控制（I/O）方式也在不断发展和完善。

无论计算机系统的规模有多大、结构有多复杂及外围设备的种类和数量有多少，从 I/O 控制的角度考虑，其逻辑上的层次体系结构基本上可以抽象成一种典型的两级模式，第 1 级：主机 - 设备接口，第 2 级：设备接口 - 外围设备，如图 5-1 所示。因此，在讨论数据 I/O 的控制时也应立足于这两级层面。

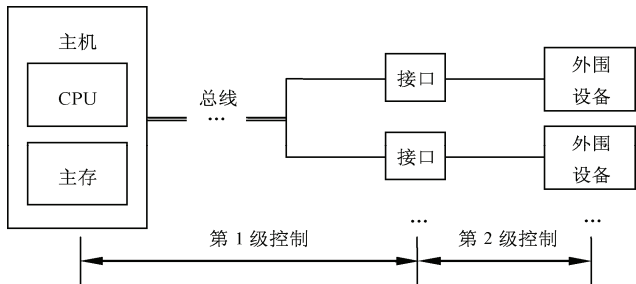


图 5-1 I/O 控制方式示意图

一般而言，在这两级层次模式下，主机与设备控制器之间通过总线实现连接和信息交互。对于简单的计算机系统，图示的总线可能就只有单独一级系统总线，如 PCI 总线，对于更复杂的大中型甚至超级计算机，图示的总线就有可能包括了若干个层次的总线，如系统总线 - 第 1 级 I/O 总线 - 第 2 级 I/O 总线等。

因此，在分析计算机系统的数 据 I/O 控制方式时，既要考虑“主机-接口”这一层面的控制机理，也要考虑“接口 - 外围设备”层面的控制机理。在计算机系统中，外围设备对主机而言，通常都已经被处理成了一个抽象存在的逻辑实体，因此在进行数据 I/O 时，对外围设备的控制一般只能直接控制设备接口，再由设备控制器根据第 1 级的控制命令来实施对外围设备的具体控制。这种典型的两级控制模式也可以理解成外围设备的 I/O 控制实际上是通过设备接口来间接实现的，即交由设备控制器来具体实施。主机则重点考虑第 1 级层面的控制，因为第 2 级控制是由设备控制器来实现的。

第 1 级控制层面的数据 I/O 传输控制通常有几种典型的方式：

1. 直接程序传输方式

直接程序传输方式（Programmed Input/Output, PIO）的工作原理是：CPU 通过执行 I/O 指令，分析设备控制器接口中专门用来指示设备运行状态的状态寄存器，了解设备当前的运行状态，再根据设备的运行状态来执行对外设的数据 I/O 操作。

PIO 方式虽然涉及的硬件结构比较简单，但在整个 I/O 操作的过程中，主机 CPU 要反复执行 I/O 指令才能完成对设备的 I/O 控制，外设的整个 I/O 操作过程都会占用 CPU，所以 CPU 不能再执行另外的任务。因此，CPU 与外设 I/O 不能并行工作，CPU 的利用率较低，I/O 的吞吐率也低，系统的响应延迟也较大。PIO 方式一般只适合于低速 I/O 设备，如传统的非增强型串口、并口、早期的 PS/2 鼠标、键盘，以及一些老旧网络接口等。现在，PIO 方式在一些功能比较单一、I/O 要求很低的单片机中还在使用。

2. 中断方式

中断方式的 I/O 控制原理是：CPU 一直执行当前分配的计算任务，当设备随机地提出了某个 I/O 请求时，CPU 立即暂停执行当前的任务，并立即切换到相应的中断服务程序执行。在中断服务程序中，CPU 进行具体的 I/O 控制，当中断服务程序执行完成后，CPU 再返回原先的计算任务继续执行。在执行中断服务程序的过程中，CPU 还可以再次响应优先级更高的设备 I/O 请求，从而实现中断嵌套。

采用中断控制方式，当设备没有 I/O 请求时，CPU 不必反复查询设备当前的运行状态，只有设备提出了 I/O 请求，CPU 才参与具体的 I/O 控制。在中断控制方式下，当没有设备提出 I/O 请求时，CPU 和设备是可以并行工作的，能把 CPU 从反复查询设备状态的简单任务中解放出来，去执行更复杂的计算任务。因此，CPU 的利用率较高，系统的响应延迟也较小，适合于中低速设备但实时性要求很高应用领域，如温度控制。

3. DMA 方式

DMA 即直接存储器访问 (Direct Memory Access)，它几乎是所有现代计算机系统都具备的一种重要 I/O 控制机制，原理上与 PIO 方式刚好相对。

DMA 是这样一种 I/O 控制方式：当设备提出了随机的 I/O 请求后，控制系统依靠控制器硬件（主要是 DMA 控制器）直接控制主机与外围设备之间的数据 I/O 操作。CPU 不参与具体的 I/O 操作控制，它只负责启动 DMA 控制器，以及执行 I/O 操作的善后处理工作。具体 I/O 操作结束以后，DMA 控制器通过中断的方式通知 CPU。

DMA 控制方式意味着主存储器与外围设备之间应有直接的数据 I/O 通路，不必经过 CPU，因此也常称为数据直传。具体就是说，设备的数据是经由总线系统直接输入到主存，而主存中的数据也是经由总线系统直接输出给设备，DMA 实际上也是因此而得名。此外，DMA 控制方式下，这种数据直传是直接由 DMA 控制器硬件来控制 I/O 操作的，不会依靠 CPU 执行指令来实现，因此在具体的 I/O 操作期间不需 CPU 参与。

用 DMA 来控制 I/O 操作，CPU 仅负责启动 DMA 控制器和执行 I/O 操作善后处理，进一步提升了主机 CPU 与外围设备的并行性，使 CPU 的利用率更高，响应时间延迟也很小，还具备高速的数据 I/O 控制能力，但其硬件机构比中断方式更复杂，一般适宜用在主存与高速外设之间的大批量数据简单传输场合，如高速磁盘的读、写等。

4. IOP 与 PPU 方式

高性能计算机中经常会用一个专用处理器来执行 I/O 程序从而实现具体 I/O 操作的控制，被称为 IOP (I/O Processor)，且因其一般位于主处理器之外，因此也被称为外围处理器 (Peripheral Processor, PP)。虽然 IOP 有自己简单的指令系统，但它的功能有限，且仍受控于主机，因此在逻辑上 IOP 仍然属于主机系统的硬件范畴。

通道是一种典型的 IOP 控制方式，其基本任务就是通过执行通道程序来管理主机与外设之间的数据输入和输出，主机 CPU 不需参与具体的 I/O 控制。通道控制器有自己的专用 I/O 控制指令，即通道指令，能够通过执行由通道指令构成的通道程序来控制多个外设的 I/O 操作，并且还能提供多个外围设备之间的 DMA 共享功能。

除 IOP 外，还有一种叫 PPU (Peripheral Processor Unit，外围处理机) 的方式。PPU 有自己完善的指令系统，能够执行多种运算和 I/O 控制，能独立于主机系统运行，因此逻辑上 PPU 可以看成是一台独立的计算机。

本章介绍的 PIO、DMA、中断及通道等在逻辑关系上并非完全相互独立。在实际系统中，这些方式之间常存在交叉融合，如中断方式下的 I/O 过程中可能涉及 PIO 方式，也可能采用 DMA 方式，在通道方式中一般还会涉及中断请求等。尽管通道的 I/O 操作一般采用 DMA，但对于一些低速设备也可用 PIO 方式。本章重点讨论 PIO、中断、DMA 和通道方式及其相关接口。

5.2 总线

总线是用来连接计算机硬件系统各功能部件，如 CPU、存储器和 I/O 设备等，并且能够被多个部件分时共享的一组信息传输的公共通道。通过总线，计算机各组成部分可以进行各种数据和命令等信息的传输。

总线有多种类型，各具特色，如有的总线连接线有 184 条（PCI 总线），有的总线连接线只有 2 条（RS-485）。通常所说的总线一般是指系统总线，它是计算机系统的重要组成部分，其性能和实现方式对整个计算机系统的功能和性能影响较大。

有的总线位于计算机的主板上，用来连接 CPU、存储器、集成电路芯片和主板上的扩展槽等；有的总线则位于机箱外，用来连接计算机的各种外设，如键盘和显示器等。一个计算机系统中通常使用多种总线来连接不同的功能部件，其要求也不尽相同。

从总线的角度看，计算机的硬件系统结构大体上有两种基本的体系模式：一种是早期的单总线模式（目前已很少使用），另一种则是现在主流的多总线模式。

1. 单总线结构

单总线结构只使用一组总线（即系统总线）连接计算机的各功能部件，其系统连接的结构特点如图 5-2 所示。单总线结构可用于连接 CPU、存储器和各类输入/输出设备。

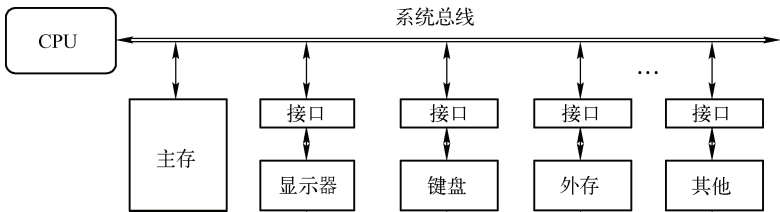


图 5-2 单总线系统的连接模式

在单总线结构中，当部件的连接特性与系统总线相一致时，可直接连接到系统总线上，如 CPU 和主存。当部件的连接特性与系统总线不一致时，需要通过相应的 I/O 接口进行连接转换，才能将该外设连接到系统总线上，如显示器、键盘和外存等设备。

2. 多总线结构

单总线计算机虽然结构简单，但总线既要兼容高速设备也要兼容低速设备，造成系统的整体 I/O 能力较差。所以，综合性能更优的多总线结构逐步发展起来，如图 5-3 所示。

多总线结构涉及多种类型的总线，如连接 CPU 与芯片组之间的 CPU 前端总线（即 Intel 平台的 FSB）、芯片组与主存之间的 PCI-E 总线、芯片组与显示器之间的 AGP 总线、连接两个芯片组之间的 DMI 总线，以及由芯片组扩展出的 PCI-E 总线或者符合特定标准的其他 I/O 总线等。

由图 5-3 可见，在同一组总线上可以同时挂接多个外围设备，如网络设备和音频设备等，而且这些外围设备可共享同一组总线，但在同一时刻只能有一个设备（即主设备）掌握总线控制权，

以实现设备之间或者该设备与主机之间的通信。

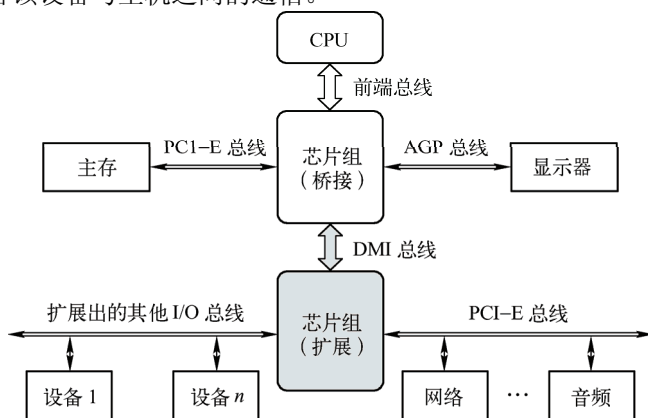


图 5-3 多总线系统的连接模式

无论是在单总线还是在多总线的计算机体系模式，总线的确是一组必不可少的信息传输公共通道，它扮演了一个必不可少的信息传输载体、路径和桥梁枢纽角色。

5.2.1 总线的特性与分类

目前，总线的类型繁多，有的总线有很高的数据传输速率，有的有较好的连接性。在信号的表示上，有的总线采用正逻辑，有的采用负逻辑。总之，不同的总线会呈现出不同的特性，其应用场合也不一样。

1. 总线的特性

由于总线是连接各部件的一组信号线。通过信号线上的信号表示信息，通过约定不同信号的先后次序即可约定操作如何实现。总线的特性如下：

- ① 物理特性。物理特性又称为机械特性，指总线上部件在物理连接时表现出的一些特性，如插头与插座的几何尺寸、形状、引脚个数及排列顺序等。
- ② 功能特性。功能特性是指每根信号线的功能，如地址总线表示地址码，数据总线表示传输的数据，控制总线表示总线上操作的命令、状态等。
- ③ 电气特性。电气特性是指每根信号线上的信号方向及表示信号有效的电平范围。通常，数据信号和地址信号定义的高电平为逻辑 1、低电平为逻辑 0，控制信号则没有统一的约定，如 \overline{WE} 表示低电平有效、Ready 表示高电平有效。不同总线的高电平、低电平的电平范围也无统一的规定，但通常与 TTL 相符。
- ④ 时间特性。时间特性又称为逻辑特性，指在总线操作过程中每根信号线上信号什么时候有效、持续多久等，通过这种时序关系约定，确保总线操作正确进行。

2. 总线的分类

总线的分类方式有多种，常见的分类方式如下。

(1) 从功能分类

总线中传输的信息主要有三类：数据信息、地址信息和控制信息，因此按所传输信息的不同，相应也有三类总线，即数据总线、地址总线和控制总线。

数据总线（Data Bus，DB）用于传输数据信号，有单向传输和双向传输数据总线之分，双向

传输时通常采用双向三态（即输入、输出和高阻三态）形式。数据总线通常由多条并行的数据线构成，如 16 条或 32 条。通常所说的 32 位总线就是指该总线的线路有 32 条。数据线越多，总线一次能够传输的位数越多。32 位的数据总线一次最多可以同时传输 32 位数据，也可以只传输 16 位或 8 位数据。

地址总线（Address Bus, AB）用于传输地址信号。地址信号由获取总线控制权的设备发出，用于选择进行数据传输的另一方或主存的存储单元。地址总线位数决定了总线可寻址范围，地址线越多寻址范围越大。例如地址总线有 20 条，则可寻址范围为 2^{20} ，即 1M 空间（ $0 \sim 2^{20}-1$ ）。

控制总线（Control Bus, CB）用于传输控制信号和时序信号。例如，Read（读）、Write（写）、Ready（就绪）等。总线的控制信号越多，总线的控制功能越强，但控制协议和操作越复杂。

一般而言，数据总线、地址总线和控制总线统称为系统总线，通常意义上所说的总线无特别说明一般是指系统总线。有的计算机系统中，数据总线和地址总线是分时分用的，即总线上在某些时刻出现的信号表示数据而另一些时刻表示地址；有的系统是分开的，51 系列单片机的地址总线、数据总线是复用的，而一般的计算机中的总线则是分开的。

（2）按数据传输格式分类

按总线所传输的数据格式，总线可以分为并行总线和串行总线。

并行总线（Parallel Bus）是用多位数据线同时传输多位，通常是 8 位（1 字节）、16 位（2 字节）或 32 位（4 字节），可同时传输的数据位数称为该总线的数据通路宽度。由于并行总线连线较多，通常只用于短距离的数据传输，其传输距离通常不超过 5 m。

串行总线（Serial Bus）一次只传输一位数据，即按数据代码位流的顺序逐位传输。串行总线的传输方式常用在传输距离较远的情况下，如外总线就常采用串行总线作为通信线路，这样可以节省硬件的成本，或者为了利用远距离的通信工具。在多机系统中，各节点之间的信息流量常低于节点内部信息流量，也常用串行总线作通信总线。

（3）按时序控制方式分类

按总线的时序控制方式分，总线可以分为同步总线和异步总线。

在同步总线（Synchronous Bus）方式中，数据传输操作由一个公共的时钟信号控制同步，这个公共时钟可以由总线控制器发出。由于采用了公共时钟，每个部件什么时候发送或接收信息都由统一的时钟规定，完成一次数据传输的时间是固定的。因此，同步通信控制简单、实现容易、且支持较高的传输频率。在同步总线中，时钟频率的设计必须兼顾到速率最慢的部件，因此时间的利用和安排不够灵活和合理，特别是部件的速率差异较大时，会降低总线效率。

在异步总线（Asynchronous Bus）方式下，其特点是没有一个统一的公共时钟，总线上的部件都可以有各自的时钟，对总线操作控制和数据传输是以应答方式实现的，操作时间根据传输的需要安排，完成一次数据传输的时间是不固定的。异步总线常用于传输距离较长、系统内各设备差异较大的场合。其优点是时间选择比较灵活、利用率高，缺点是控制比较复杂。

常见的微型计算机系统中较多采用同步总线，但部分引入了异步控制思想，如 PC 总线。具体方法是让总线周期所含时钟数可变，如果时钟频率较高（时钟周期短），则时间浪费就变得很小，总线周期的长度可以根据需要灵活调整。也可以采取请求—批准—释放的应答方式，但以时钟周期为时间的基准。这既保持了同步总线的优点，也在一定程度上具有异步总线的优点。

有的系统总线，如 IBM PS/2 中的微通道总线，具有几种选择：同步周期（标准周期）方式、扩展同步周期方式、异步方式，既可支持同步方式，也可支持异步方式。

(4) 按总线的结构层次分类

按结构层次进行划分, 总线可分为芯片内总线、局部总线、系统总线和外部总线等。

芯片内总线是指用于连接芯片内部各基本逻辑单元的总线, 如 CPU 的内总线等。

局部总线用来连接 CPU 和外围控制芯片和功能部件, 用于芯片一级的互连, 有时也被称为局部总线, 如 Intel 平台的 FSB (即前端总线)、AMD 平台的 HT 总线等。

CPU 一般通过存储总线 (Memory Bus) 与主存或者高速显卡进行数据传输, 主要有数据线、地址线和控制线。从结构层次上看, 存储总线也属于芯片一级的总线, 因此它一般被归属为局部总线。

系统总线 (System Bus) 也称为 I/O 通道总线, 是用来与扩展插槽上的各扩展板相连的总线, 用于部件一级的互连, 通过它将各部件连接成一个计算机系统。通常所说的总线一般是专指系统总线, 如 ISA、PCI 及 AGP 总线等。

外部总线 (External Bus) 是若干计算机系统之间或计算机系统与其他系统 (如通信设备、传感器等) 之间的连接总线, 有时也被称为通信总线, 常用于设备一级的互连。例如, 连接并行打印机的 Centronics 总线、USB 总线和 CAN 总线等。

5.2.2 总线的标准

总线的主要功能就是连接计算机系统各个功能部件, 为了使不同的功能部件和接口都能连接到总线上并与之交互通信, 就必须详细定义总线的各项技术规范, 即总线标准。任何一个部件只要符合总线标准都可以连接到总线上, 并与总线上的其他设备进行数据交换。总线标准除了定义信号线的功能外, 还必须制定总线的机械和电气方面的规格, 使负载适宜, 接头合适, 并能提供合适的电压和时序信号等。

每个总线标准都有详细的规范说明, 它们通常是一个包含上百页、几十万字 (含大量图标) 的文档。总线标准主要包括以下几部分:

① 机械结构规范——确定模块的尺寸、总线插头、边沿连接器插座等规格及位置等。

② 功能规范——确定总线每根线 (引脚) 信号的名称和功能, 对它们相互作用的协议 (如定时关系) 和工作过程等进行说明。

③ 电气规范——规定总线每根信号线在工作时的有效电平、动态转换时间、负载能力及各类电器性能的额定值、最大值等。

为了使不同的外部设备都能连接到总线上, 使外部设备的开发独立于主机和系统总线, 制定总线标准就显得尤其重要。总线标准为计算机系统中各模块的互连提供了一个标准的界面, 该界面对其连接在两侧的模块部件都是透明的, 界面任一方只要根据总线标准的要求来实现接口的功能, 就可实现各种配件的插卡化。

特别是在微型计算机系统中, 系统组成模式灵活多样。一类是选用某一主流微机系统, 再根据自己的需要扩充功能插件。另一类是选用有关的功能模块 (插件), 再按照自己的需要, 积木式地组装成系统。这就更需要约定某种互连标准 (即总线标准), 包括上述机械结构、功能和电气方面的规范。制定了总线标准以后, 用户就可以自行选购符合某种总线标准的部件, 方便对计算机硬件系统进行扩展, 甚至不必关心插件的内部细节。

国际上制定总线标准的组织主要有 IEEE (Institute of Electrical and Electronics Engineers, 电气和电子工程师协会)、ITU (International Telecommunication Union, 国际电信联盟)、ANSI (American National Standards Institute, 美国国家标准学会)、EIA (Electronic Industries Association,

电子工业协会)和 PCI-SIG (Peripheral Component Interconnect Special Interest Group, 外围部件互连特别兴趣组)。目前, 计算机系统中常用的总线标准主要有以下几种。

(1) ISA (Industry Standard Architecture, 工业标准结构) 总线

ISA 是 IBM 公司于 1984 年为 PC/AT 机制定的系统总线标准, 又称为 AT 总线。

ISA 总线共有 98 条引脚线, 为了与早期的 XT 总线兼容, ISA 总线的扩展槽由两部分构成: 一部分由 62 引脚构成, 其信号分布及名称与 PC/XT 总线的扩展槽基本相同, 仅有很小的差异。另一部分是 AT 机的添加部分, 由 36 引脚组成。这 36 引脚分成两列, 分别称为 C 列和 D 列。

ISA 总线是 8/16 位总线, 当只使用 ISA 总线插槽的前面部分 (由 62 引脚构成) 时, 与 XT 总线兼容, 有 20 条地址线, 8 条数据线, 作为一个 8 位总线使用; 当两部分都使用时, 可作为一个 16 位总线使用, 地址线也增加到 24 位, 可寻址 16 MB 地址空间。ISA 总线的主频为 8.33 MHz, 因此 ISA 总线的最大数据传输率能达到 8.33/16.66 MB/s。

ISA 总线曾在 80286 到 80486 的计算机中广泛使用。

(2) EISA (Extended Industry Standard Architecture, 扩展工业标准结构) 总线

EISA 总线是由 Compaq 等 9 家公司于 1988 年联合推出的总线标准。EISA 总线是 32 位总线, 是为满足 32 位微处理器需求而推出的。EISA 总线与 ISA 总线兼容, 这样可以保护厂商和用户已在 ISA 总线上的巨大软硬件投资。EISA 总线的引脚数有 196 条, EISA 总线的连接器是一个双层插槽设计, 既能接受 ISA 卡, 又能接受 EISA 卡, 上层与 ISA 卡相连, 下层则与 EISA 卡相连。

EISA 总线主要性能指标包括: ① 32 位地址线, 可直接寻址范围 $2^{32}=4\text{G}$; ② 32 位数据线; ③ 总线时钟频率 8.33 MHz, 最大数据传输率 33.3 MB/s。

(3) AGP (Accelerated Graphics Port, 加速图像接口) 总线

AGP 是 Intel 公司推出的一种 3D 标准图像接口, 能够提供 4 倍于 PCI 的效率。

随着多媒体的深入应用, 三维技术的应用越来越广, 处理三维数据不仅要求有惊人的数据量, 还要求有更宽广的数据传输带宽, PCI 总线已不能满足快速数据传输的需求。为了解决此问题, Intel 于 1996 年 7 月推出了 AGP 总线, 这是显示卡专用的局部总线, 基于 PCI 2.1 版规范并进行扩充修改而成, 以 66.6 MHz 的频率工作, 采用点对点通信方式, 允许 3D 图形数据直接通过 AGP 总线进行传输。

AGP 总线的主要特点包括: 以主存作为帧缓冲器, 即将原来存于帧缓冲区中的纹理数据存入主存中, 采用流水线操作, 从而减少了内存的等待时间, 提高了数据传输速率。

AGP 总线的数据线是 32 位, 有多种工作方式: 基频工作 (以 66.6 MHz 的频率工作)、2 倍频工作、4 倍频工作和 8 倍频工作, 对应的数据传输速率分别是 266.4 MB/s、532.8 MB/s、1065.6 MB/s 和 2131.2 MB/s。

(4) PCI 和 PCI-Express

关于 PCI 和 PCI-Express 的详细情况请参见 5.2.4 节的相关内容。

5.2.3 总线的设计要素

在遵守某种总线标准的前提下, 无论哪一种总线, 其设计要素都会涉及总线的位宽、工作频率、带宽、时序控制及仲裁方式等技术层面。下面将从总线的技术指标、时序控制和仲裁方式这三方面来分析总线的设计要素。

1. 总线的技术指标

(1) 总线宽度

总线的位宽又称为总线的宽度，是指总线中数据线的位数。例如，总线的宽度若为 32 位，则表明通过该总线进行一次数据传输最多可以传输 32 位，当然也可以少于 32 位，如每次只传输 16 位或者 8 位。

显然，增加总线的宽度能提高总线的数据传输率。例如，如果数据总线有 8 位，每条指令长 16 位，那么每个指令周期必须访问存储器 2 次才能读出指令。如果数据总线为 16 位，则只需访存一次就能读出该条指令。

地址总线用于传输读入或写出数据所在单元的地址。地址线的宽度决定了计算机系统能够使用的最大寻址空间。总线中地址信号线数越多，CPU 能够直接寻址的内存空间也就越大。若总线中有 n 位地址线，则 CPU 能用它对 2^n 个不同的内存单元进行寻址 ($0 \sim 2^n - 1$)。为达到更大的寻址空间，总线需要的地址线就应越多。

总体而言，总线宽度越宽，越能提高系统的性能。但随着连接线数的增加，系统需要更大的物理空间（如主板上的总线要占用更多的面积）和更大的连接器，这些因素都将增加总线的成本。

为平衡性能与成本的问题，现代计算机系统常用复用总线设计代替原来的分立专用总线设计。在分立专用总线方式中，地址线 and 数据线是分开的。由于在数据传输的开始，总是先把地址信号放在总线上，等地址信号有效后，才能开始读写数据。这样地址和数据信息都可以用同一组信号线传输，在总线操作开始时，这些线路传输地址信号，随后它们又可以继续传输数据信号。分时复用的优点是减少了总线的连线数，从而降低了成本，节约了空间。其缺点是降低了系统的速度，增加了连接和控制的复杂度。

(2) 总线频率

总线工作速度的一个重要指标就是总线频率，是指总线每秒进行传输数据的次数。总线频率越高，则总线的数据传输率越高。总线频率通常用 MHz 表示，如 PCI 总线标准频率为 33 MHz，PCI-E 1.0 总线的标准频率为 2.5 GHz。

(3) 总线带宽与数据传输率

在计算机中，“带宽”一词通常已被借用来表示总线的数据传输率。因此总线的带宽也是指单位时间内总线上的数据传输量，单位 b/s（比特/秒）。总线的实际带宽与总线的位宽和总线的频率及编码方式等参数相关，实际带宽与各参数之间的基本关系是：

$$BW = \frac{f \times w \times d \times L \times E}{8} \quad (\text{Bps}) \quad (5-1)$$

其中，BW 是总线的实际带宽， f 是总线的工作频率， w 是总线的位宽， d 是工作模式（单工 $d=1$ ，双工 $d=2$ ）， L 是总线的通道（Lane）路数， E 是编码方式。对于单工单通道的总线，如标准的 PCI 总线，其 $d=1$ 、 $L=1$ ，若忽略 E ，则式（5-1）与式（1-1）的含义一致。

例如，PCI-Express 3.0 X32 的总线频率为 8 GHz，1 bit 宽，双工模式，采用 128/130 编码方式，则它的最大实际带宽 $BW = (8 \text{ GHz} \times 1 \text{ bit} \times 2 \times 32 \times 128/130) \div 8 \approx 63.02 \text{ GB/s}$ 。

2. 总线周期与操作过程

总线周期通常指的是 CPU 完成一次访问主存或 I/O 端口操作所需要的时间。总线的操作过程，是指完成两个设备之间信息传输的完整过程。这里的设备主要是指主存和 I/O 端口。在通常情况下，一个总线周期与一次操作过程是对应的。

申请并掌握总线控制权的一方称为主设备，另一方则称为从设备。

注意：区别主设备与从设备身份差异的基本依据在于谁申请并掌握总线控制权，并不在于谁发送数据、谁接收数据，主设备可以发送数据也可以接收数据，反之，对从设备也一样。

总线操作的步骤可以概括为：

- (1) 主设备申请总线控制权，总线控制器进行裁决并发出批准信号。
- (2) 主设备掌握总线控制权，启动总线周期，发出地址码和总线操作类型。
- (3) 从设备响应，主—从设备之间进行数据传输。
- (4) 主设备释放控制权，结束总线周期。

一个总线周期具体持续多长时间，与系统的时钟周期及总线采用的控制方式有关。在同步控制方式下，一个总线周期一般包括 4 个基本的时钟周期，有时会插入一些延长的时钟周期。插入的延长时钟周期数为 0，即为标准的同步控制方式；插入的延长时钟周期数不为 0，则为扩展的同步控制方式。在异步控制方式下，一个总线周期的时间就只能视具体情况而定，无统一的标准。

总线上的数据传输模式主要有如下两种。

- ① 单周期模式：一个总线周期只传输一个数据，传输完成后主设备释放总线控制权。如果还需再次传输数据，则应重新申请总线控制权。
- ② 突发模式 (burst)：主设备获取总线控制权后，可以进行多个数据的传输。在总线周期中寻址时，只给出目的首地址，访问数据 1、数据 2、数据 3、…、数据 n 时的地址在首地址基础上按一定规则自动产生（如自动加 1）。

3. 总线的控制方式

总线的数据传输操作是在时序信号的控制下进行的。如前所述，根据时序控制的方式不同，总线可划分为同步控制总线和异步控制总线。

(1) 同步控制方式

同步控制方式的主要特征是以时钟周期为划分时间段的基准。总线的每一步操作都必须在规定的时间段内完成，每次数据传输所需的时间段数是固定的。

由于外设的速度通常较慢，而且变化幅度也比较大，因此在实际使用的各类同步总线中，往往允许每次数据传输所需的时间段数是可变的，以适应不同操作速度的需要。改进途径主要有两种：一是同步控制方式的数据读/写过程中插入若干延长周期，二是在同步控制方式中引入以时钟周期为基础的“请求—应答”方式。第二种改进途径可以看成在同步控制的基础之上部分地引入了异步控制思想，时间能长则长，能短则短，由设备根据实际情况协商决定，但这种方式还是以基本的时钟周期为基准，因此称为扩展的同步控制方式。

图 5-4 是一个总线同步控制方式的时序示意图。

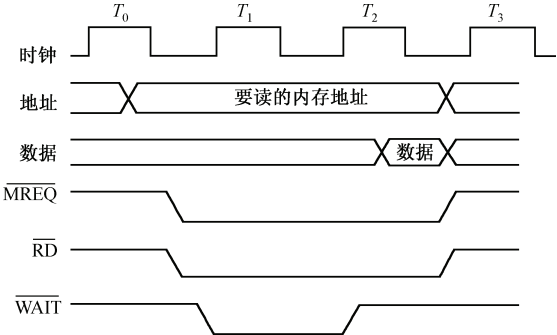


图 5-4 同步总线的读时序

在 T_0 时, CPU 在地址总线上给出要读的内存单元地址, 在地址信号稳定后, CPU 发出内存请求信号 $\overline{\text{MREQ}}$ 和读信号 $\overline{\text{RD}}$ 。信号 $\overline{\text{MREQ}}$ 说明 CPU 要访问的是主存储器单元 (反之则是访问外设), 信号 $\overline{\text{RD}}$ 说明要进行读操作。

由于内存芯片的读写速率低于 CPU, 在地址建立后不能立即给出数据, 因此内存存在 T_1 的起始处发出一等待信号 $\overline{\text{WAIT}}$, 通知 CPU 插入一个等待周期, 直到内存完成数据输出并将 $\overline{\text{WAIT}}$ 信号置反, 等待周期的插入可以是多个。

在 T_2 的前半部分, 内存将读出的数据放到数据总线, 在 T_2 的下降沿, CPU 选通数据信号线, 将读出的数据存放至内部寄存器中。读完数据后, CPU 再将 $\overline{\text{MREQ}}$ 和 $\overline{\text{RD}}$ 信号置反。如果需要, CPU 可以在时钟的下一个上升沿启动另外一个访问内存的周期。

图 5-5 是以时序状态图的形式, 简要描述了 DMA 控制器掌握总线, 且在同步控制方式下的一次总线操作过程。它表明了 DMA 控制器向 CPU 提出总线请求, 在获得总线控制权后通过总线完成了一次数据传输。

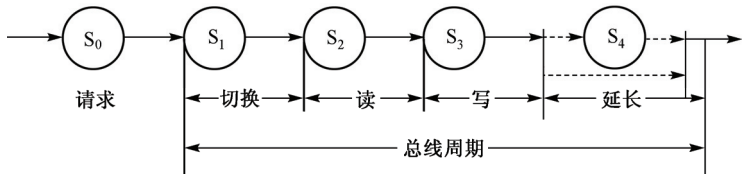


图 5-5 同步总线的操作状态图

① S_0 状态。DMA 控制器提出总线请求, 送往 CPU。此时 CPU 可能正在控制系统总线去访问主存, 因此 DMA 控制器就只能处于等待总线请求被批准的 S_0 状态, 也有可能需要等待几个时钟周期。

② S_1 状态。CPU 结束一次总线周期操作后, 发出总线批准信号, 然后进入总线控制权交换状态 S_1 。在 S_1 中, CPU 对总线的有关输出呈高阻态, 与总线脱钩 (即放弃总线)。DMA 控制器向总线送出地址码, 接管总线控制权, 并进入 S_2 状态。一般, S_1 只需一个时钟周期即可完成总线控制权的切换。

③ S_2 状态。在 S_2 中, 由 DMA 控制器发出读命令, 从发送设备中读出数据, 并送入有关的接口数据寄存器, 然后再发送到数据总线上。

④ S_3 状态。在 S_3 中, 由 DMA 控制器发出写命令, 将数据总线信息写入接收设备。

⑤ 延长状态 S_4 。如果在 S_2 或 S_3 中没有完成总线传输, 则可延长总线周期, 进入 S_4 , 继续总线传输操作。 S_4 可以是一个时钟周期或数个时钟周期。

结束一个总线周期后, DMA 控制器放弃总线控制权 (有关输出呈高阻态), 并将总线的控制权交回 CPU。在图 5-5 中, DMA 控制器所掌管的一个总线周期包括 S_1 、 S_2 、 S_3 , 或许还有 S_4 。其中, S_1 用于接管总线, S_2 和 S_3 等用于实现总线传输操作。 S_0 属于前一个总线周期。在同步方式中, 总线周期的宽度一般为时钟周期宽度的整数倍。

(2) 异步控制方式

异步控制方式没有统一的时序信号来控制各项操作, 设备之间采取交互式“请求-应答”的方式来实现通过总线的数据传输控制, 所需总线时间视需要而定。图 5-6 给出了异步总线上执行的一次读操作过程的示意图。这里, 我们可以将申请并获得总线控制权的设备称为主设备, 将响应主设备请求并与之通信的设备称为从设备。

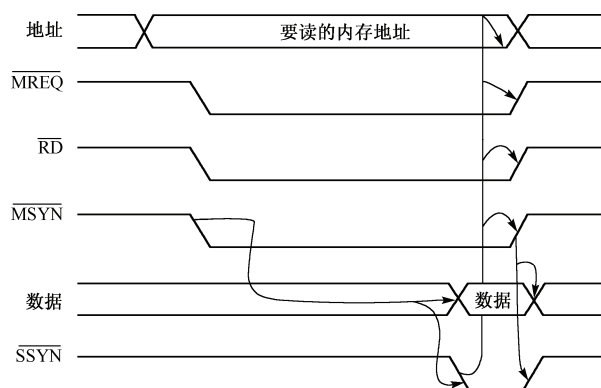


图 5-6 异步总线读操作时序

在异步操作中，主设备在给出地址信号、主存请求信号 $\overline{\text{MREQ}}$ 和读信号 $\overline{\text{RD}}$ 后，再发出主同步信号 $\overline{\text{MSYN}}$ ，表示有效地址和控制信号已送上系统总线。从设备得到这个信号后，以其最快的速度响应和运行，完成所要求的操作后，发出从同步信号 $\overline{\text{SSYN}}$ 。

主设备得到从同步信号 $\overline{\text{SSYN}}$ ，就知道数据已经就绪，且已出现在数据总线上，从而接收数据，并撤销地址信号，将 $\overline{\text{MREQ}}$ 、 $\overline{\text{RD}}$ 和 $\overline{\text{MSYN}}$ 信号置反。从设备检测到 $\overline{\text{MSYN}}$ 信号已置反后，得知主设备已接收到数据，一个访问周期已经完成，因此将 $\overline{\text{SSYN}}$ 信号置反。这样就回到了起始状态，又可以开始下一个总线周期。

异步总线操作时序图中的箭头表示了事件起始和结束的因果关系，即代表了异步应答信号的关系。一般将异步应答关系分为不互锁、半互锁、全互锁三类，如图 5-7 所示。

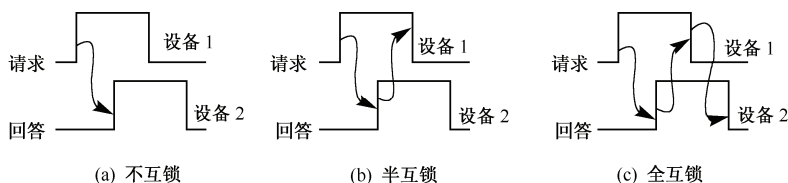


图 5-7 请求与回答信号的互锁

在图 5-7(a)中，设备 1 发出请求信号，经过一定时间（确信设备 2 收到请求信号）后，自动降下请求信号。设备 2 收到请求后，在条件允许时发出回答信号，经过一定时间（确信回答信号发生作用了）后，自动撤销回答信号。在这种应答方式中，回答信号是因请求信号而引发的，用箭头表示这种引发关系。但两个信号的结束都是由设备自身定时决定的，不存在互锁关系，因此称为不互锁方式。

在图 5-7(b)中，设备 1 在接收到设备 2 的回答信号后，知道它的请求信号已被接收，便撤销请求信号。但回答信号的撤销则仍由设备 2 本身定时决定，不采取互锁方式来控制。因此，这种方式称为半互锁方式。

在图 5-7(c)中，设备 1 在接收到设备 2 的回答信号后，撤销其请求信号。设备 2 在获知请求信号撤销后，便撤销其回答信号。因此，这种方式称为全互锁方式。

采用全互锁方式，一旦完成任务便立即撤销有关信号，时间安排非常紧凑；但全互锁方式的实现比较复杂，可靠性会相应地降低。采用非互锁方式，按照固定定时结束信号，较易实现，可靠性较高；但信号维持时间必须满足最长操作的需要，对耗时更短的操作则会出现时间浪费现象，可能降低总线操作的效率。

图 5-6 所示的异步操作时序即为一种全互锁方式，如 $\overline{\text{MSYN}}$ 信号的给出使数据信号建立，并使从设备发出 $\overline{\text{SSYN}}$ 信号。反过来， $\overline{\text{SSYN}}$ 信号的发出将导致地址信号的撤销以及 $\overline{\text{MREQ}}$ 、 $\overline{\text{RD}}$ 和 $\overline{\text{MSYN}}$ 信号置反。最后， $\overline{\text{MSYN}}$ 信号的置反导致 $\overline{\text{SSYN}}$ 信号置反，结束整个操作过程。

总线的同步时序控制方式实现和控制都很简单，但是时间利用不合理，也没有异步时序灵活；同时，一旦总线的周期确定了，以后即使出现了更快的设备，也无法发挥其性能。异步时序的总线不但能合理利用时间，而且不论设备是快还是慢，使用的技术是新还是旧都可以共享总线，但异步总线控制比较复杂。采用以时钟周期为基准但引入了异步控制的思想的扩展同步总线方式在一定程度上兼有两者的优点，现在应用比较广泛。

4. 总线的仲裁

在总线上连接了多个部件，任意两个部件间都可以通过总线传输数据，这样在某一时刻就有可能出现不止一个部件提出使用总线申请。当多个部件同时申请使用总线时，就会出现总线冲突的现象，因此必须采用一定的方法对总线的使用权进行仲裁。按照总线仲裁电路的位置不同，仲裁方式可分为集中式仲裁和分布式仲裁两种。

(1) 集中式总线仲裁

集中式总线仲裁需要中央仲裁器，总线控制逻辑基本上集中放在一起，分为链式查询方式、计数器定时查询方式和独立请求方式。在集中式总线仲裁方式中，由专门的总线控制器或仲裁器来管理总线的使用，总线控制器可以包含在 CPU 内，但更多的是由专门的器件来承担的。连接在总线上的设备都可以发出总线请求信号，当总线控制器检测到有总线请求时，它发出一个总线授权信号。

链式查询方式如图 5-8 所示，即总线授权信号被依次串行地传输到所连接的 I/O 设备上。当逻辑上离控制器最近的那个设备接收到授权信号时，如果该设备发出了总线请求信号，则由它接管总线，升起总线忙信号，并停止授权信号继续往下传播。若该设备没有发出总线请求，则将授权信号继续传输到下一个设备。这个设备再重复上述过程，直到有一个设备接管总线为止。在链式查询方式中，设备使用总线的优先级由它离总线控制器的逻辑距离决定，越近的优先级越高。

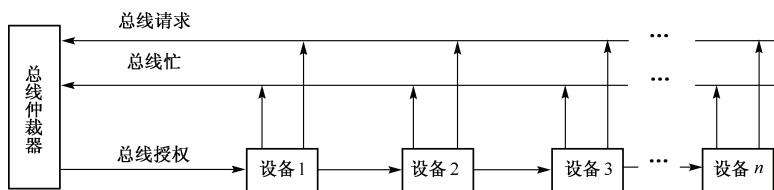


图 5-8 链式查询方式总线仲裁

链式查询方式的优点是查询链路简单，易于控制和扩充新加设备，缺点主要是对故障比较敏感，当某个设备出现故障时，会中断查询链路，无法继续传递总线授权信号。此外，链式查询方式中各设备的优先级按与总线仲裁器的距离大小，越近的优先级越高，因此优先级是固定的。

计数器定时查询方式如图 5-9 所示。总线上的每个部件通过“总线请求”线发出请求信号，总线仲裁器（内设查询计数器）收到请求以后，计数器开始计数，定时查询各设备以确定是哪个设备发出的请求。当查询计数器的计数值与发出请求的设备编号一致时，计数器中止查询，该设备发出总线忙信号后获得总线控制权。

启动每次新的计数前，计数器清 0，查询则从 0 开始进行，设备优先级安排就类似于链式查询方式。如果每次查询前计数器不清 0，则从中止点继续查询，就是一种循环优先级，为所有设

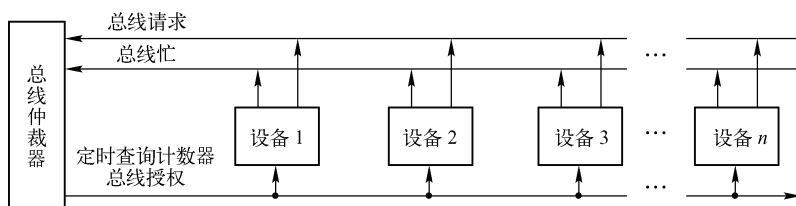


图 5-9 计数器定时查询方式总线仲裁

备提供相同的总线使用机会。如果启动新查询时将计数器设置为某个初值，则可以设定某个设备为最高优先级。

计数器定时查询方式的总线仲裁优点是优先级比较灵活，计数初值、设备编号都可以通程序设定，优先次序可用程序来控制。某些设备故障也不会影响到其他部件，可靠性较高。其缺点是需要向各设备广播计数值，因此会增加连线数量，控制复杂。

独立请求方式如图 5-10 所示。每个设备都通过自己专用的总线请求信号线与总线仲裁器连接，并通过独立的总线授权信号线接收总线批准信号。需要使用总线时，各设备独立地向总线仲裁器发送总线请求信号，仲裁器可以根据某种算法对同时送来的多个总线请求进行仲裁，以确定批准哪个设备可以使用总线，并通过该设备的总线授权线向设备发送总线批准信号，该设备再通过公共线路设置总线忙信号后获得总线的控制权。

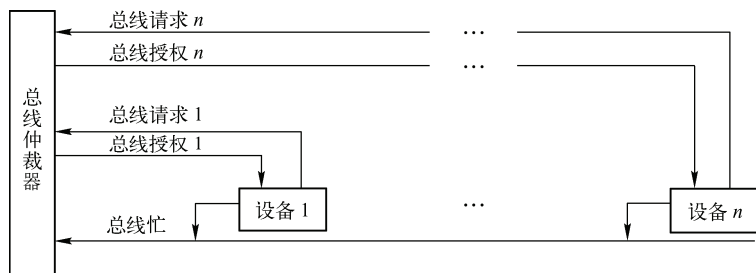


图 5-10 独立请求方式总线仲裁

独立请求方式的优点是总线分配速度快，所有设备的总线请求信号能同时送到总线仲裁器，不需要进行任何查询。仲裁器可以使用多种方式由程序灵活地设置设备的优先级，也能方便地隔离故障设备的总线请求。其缺点是控制线路数量较大，所需要的独立线路较多，成本较高、控制起来复杂。

(2) 分布式总线仲裁

与集中式仲裁方式相比，分布式仲裁不需要设置一个集中的总线仲裁器，各设备都设有自己专用的仲裁电路和设备号，各设备之间通过仲裁电路竞争使用总线，如图 5-11 所示。

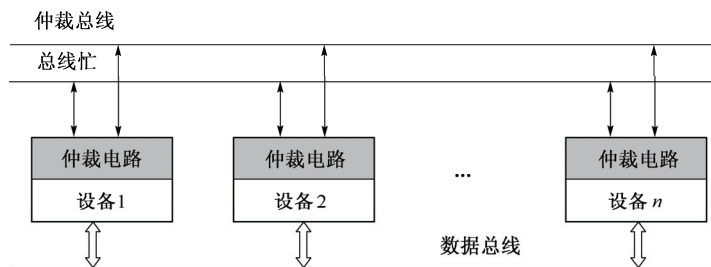


图 5-11 分布式总线仲裁

分布式仲裁方式以优先级仲裁策略为基础，当设备有总线请求时，就把各设备唯一的设备号发送到共享的仲裁总线上，由各设备自己的仲裁电路去比较，通过比较保留设备号大的设备、撤销设备号小的设备，竞争获胜者的设备号将保留在仲裁总线上，设备通过设置总线忙信号而获得总线的控制权，并在下一个总线周期使用总线。

与集中式总线仲裁相比，这种总线仲裁方式要求的总线信号更多、控制电路也更复杂，但它能有效地防止总线仲裁过程中可能出现的时间浪费，提高总线的仲裁速度。

5.2.4 PCI 总线介绍

PCI (Peripheral Component Interconnect) 总线，即外部组件互连总线，最早是由 Intel 公司于 1991 年提出的高带宽、独立于处理器的总线。PCI 总线能够作为中间层或直接连接外围设备的总线，允许在计算机内安装多达 10 多个遵从 PCI 标准的扩展卡。PCI 总线取代了早先的 ISA 总线，它有许多优点，如能够支持即插即用 (Plug and Play) 和中断共享等。

1992 年，Intel 正式发表 PCI 1.0 标准，但该标准仅限于组件级规范。1993 年，PCI-SIG (Special Interest Group) 又发表了 PCI 2.0 标准，此标准建立了连接器与主板插槽间的连接规范。PCI 标准一经发表就被立即应用于服务器中，取代了先前的 MCA 及 EISA，成为服务器扩展总线的不二选择。在主流个人计算机中，PCI 标准则缓慢地取代着 VESA 局部总线 (VLB)，直至 1994 年后期第二代奔腾 PC 推出后，该标准才得以实现具有重要意义的市场突破。1996 年，VLB 彻底退出了个人计算机市场，制造厂商甚至将 PCI 标准应用于 486 计算机中。EISA 标准则与 PCI 标准共存使用至 2000 年。苹果公司在 1995 年中期将 PCI 标准应用于专业计算机 Power Macintosh 中 (取代了 NuBus)，普通产品 Macintosh Performa 则于 1996 年中期才完成了换代 (取代了 LC PDS)。

PCI 标准的后续版本不断加入了新的功能与性能提升，如包括 66 MHz/3.3 V 标准、133 MHz PCI-X 及适应多种主板板型等。2002 年，PCI 的串行版本 PCI Express 问世后，主板制造商逐渐减少了传统 PCI 插槽，大量引入了 PCI Express 接口。

最早的 PCI 总线的频率为 33 MHz，每个总线周期可并行传输 32 位数据，总线的带宽约为 133 MBps。在 1993 年，PCI 总线的数据宽度发展到 64 位，总线频率也提高到了 66 MHz。到目前为止，PCI 已发展到了 PCI-E 3.0 (2010 年由 PCI-SIG 正式发布) 阶段，能支持高达 2.5/5/8 GHz 的总线频率，采用 128/130 编码方式，最高可支持 32 个双工通道 X32，其最大带宽可达 64 GBps。

PCI 总线可直接连接高速外部设备 (如图形显示适配器、网络接口、SCSI 设备等)，为其提供了更好的性能。其次，PCI 总线支持把其他总线连到 PCI 总线上。在图 5-3 中，通过 ISA 桥接器，可将 PCI 总线和 ISA 总线连接在一起，而且该桥接器还能支持一到两个 IDE 盘。PCI 桥连接了 CPU、内存和 PCI 总线。这种结构的最大优点是 CPU 与内存之间的带宽特别高，有了这条专用的高速内存总线后，PCI 总线就可以为快速的外部设备提供较高的带宽，而且原来的 I/O 总线 (如 ISA) 能照常使用。

PCI 是一种同步时序总线，对地址信号和数据信号进行了复用，支持 64 位地址和 64 位数据信息，采用集中式总线仲裁方式，支持 5 V 和 3 V 两种电源电压，支持 32 位和 64 位扩展卡并向后兼容，总线规范独立于微处理器，通用性较好，其配置支持单个和多个处理器系统。PCI 总线是专门为满足现代计算机系统的 I/O 要求而设计的较经济的总线，实现 PCI 总线只需很少的芯片。

1. 信号组成

PCI 信号可分为必备和可选两大类。PCI 总线向主设备提供必备信号为 49 个，向从设备提供

必备信号为 47 个，可选信号为 51 个，主要用于 64 位扩展、中断请求和高速缓存支持等，利用这些信号可以处理数据、地址信息，实现接口控制、仲裁及系统功能。综合起来，不包括电源、接地和保留线等，PCI 一共提供 50 个必备信号和 50 个可选信号。

(1) 必备信号

- ① 系统信号：包括时钟和复位线。
- ② 地址和数据信号线：32 根分时复用的地址数据线。
- ③ 接口控制信号线：控制数据交换的时序并提供主设备和从设备的协调。
- ④ 仲裁信号线：仲裁信号线是非共享的线，每个 PCI 部件都有它自己的一对仲裁线，直接连接到 PCI 总线仲裁器上。
- ⑤ 错误报告信号线：用于报告奇偶校验错误和其他类型的错误。

表 5-1 给出了必备信号的说明，其中信号类型和符号含义如下：

表 5-1 PCI 必备信号说明

| 信号名称 | 信号线数 | 类型 | 描 述 |
|----------------|------|-------|---|
| 系统信号 | | | |
| CLK | 1 | IN | 系统时钟信号。33 MHz 或 66 MHz，在上升沿被所有的输入所采样 |
| RST# | 1 | IN | 复位信号。强迫所有 PCI 专用的寄存器、定序器和信号复位为初始化状态 |
| 地址和数据信号 | | | |
| AD[31:0] | 32 | T/S | 复用的地址/数据信号线 |
| C/BE[3:0] # | 4 | T/S | 复用的总线命令和字节选定信号：送地址期间，定义总线命令；在数据期，表示 32 位的 4 字节通路中的哪一个是有意义的的数据。使用该信号后，就可以读写一个整字或其中的 1、2、3 字节 |
| PAR | 1 | T/S | 地址或数据的校验位 |
| 接口控制信号 | | | |
| FRAME# | 1 | S/T/S | 周期帧信号：由当前主设备驱动，表示交换的开始和持续的时间，即指明 AD 和 C/BE 信号已发出 |
| IRDY# | 1 | S/T/S | 当前主设备准备好信号：读操作时，表示主设备准备好接收数据；写操作时，表示数据已出现在总线上 |
| TRDY# | 1 | S/T/S | 从设备就绪：读操作时，有效数据已在总线上；写操作时，从设备准备好接收数据 |
| STOP# | 1 | S/T/S | 停止信号：从设备需要停止当前的传输 |
| LOCK# | 1 | S/T/S | 锁定信号：表示一个操作可能需要多个传输周期（且中途不能中断）才能完成 |
| IDSEL | 1 | IN | 初始化设备选择：通参数配置读写操作期间的芯片选择 |
| DEVSEL# | 1 | IN | 设备选择：由当前被选中的从设备驱动；信号有效时，说明总线上有某个设备被选中 |
| 总线仲裁信号 | | | |
| REQ# | 1 | T/S | 总线仲裁信号：向总线仲裁器申请总线使用权 |
| GNT# | 1 | T/S | 总线仲裁响应信号：向设备指明仲裁器允许其使用总线，这是一条各设备都有的专用的点对点信号线 |
| 错误报告信号 | | | |
| PERR# | 1 | S/T/S | 奇偶校验错：在数据传输时，表示检测到数据校验有错 |
| SERR# | 1 | O/D | 系统错误：用于报告地址奇偶校验错和其他系统错误 |

- ⊙ IN：单向输入信号。
- ⊙ OUT：单向输出信号。
- ⊙ T/S：双向三态输入/输出信号。
- ⊙ S/T/S：持续低电平有效的三态信号，一次只有一个拥有者驱动的持续三态信号，驱动 S/T/S 信号必须在它悬浮之前维持一个时钟的高电平；新的驱动信号必须在三态之后一个时钟才开始驱动。
- ⊙ O/D：漏极开路，允许多个设备以“线或”的形式共享该信号。

◎ #: 低电平有效。

(2) 可选信号

① 中断请求信号 INTX#: 共 4 位, O/D, 用于中断请求, X=A、B、C、D, 其中 B、C、D 只对多功能设备有意义。

② 高速缓存支持信号, 包括:

◎ SBO#——1 位信号线, IN/OUT, 测试返回, 信号有效时, 表示监听命中高速缓存的修改行 (针对多处理器)。

◎ SDONE——1 位信号线, IN/OUT, 监听完成, 指示当前监听状态 (针对多处理器)。

③ 64 位总线扩展信号, 包括:

◎ AD[63::32]——32 位信号线, T/S, 将总线扩展为 64 位地址/数据复用线。

◎ C/BE[7::4]#——4 位信号线, T/S, 字节选定的另外 4 位。

◎ REQ64#——1 位信号线, S/T/S, 64 位传输请求信号。

◎ ACK64#——1 位信号线, S/T/S, 64 位传输允许信号。

◎ PAR64——1 位信号线, T/S, 附加的 32 位地址/数据的校验位。

④ TAG 边界扫描信号, 包括:

◎ TCK——1 位信号线, IN, 测试时钟。

◎ TDI——1 位信号线, IN, 测试输入。

◎ TDO——1 位信号线, OUT, 测试输出。

◎ TMS——1 位信号线, IN, 测试模式选择。

◎ RST#——1 位信号线, IN, 测试复位。

2. 总线仲裁

PCI 总线使用的是集中式总线仲裁器, 独立请求方式。如图 5-12 所示, 每个 PCI 设备都有独立的请求 (REQ) 和允许 (GNT) 线连接到 PCI 仲裁器。仲裁器一般都设置在某个搭桥芯片上。REQ#信号线用于设备发出总线请求, GNT#信号线用于接收总线授权。

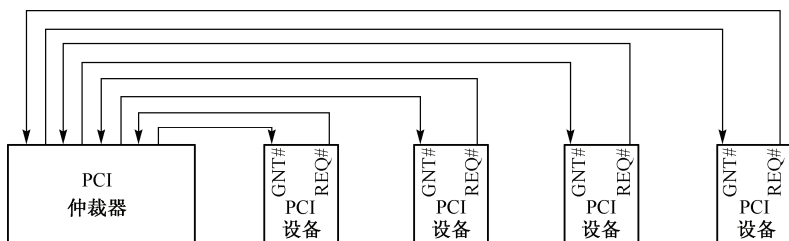


图 5-12 PCI 总线仲裁示意图

为了申请总线的使用权, 首先由 PCI 设备(主设备, 包括 CPU 在内)发出总线请求信号 REQ#, 并等待总线仲裁器的响应信号 GNT#。只有收到响应信号的设备, 即得到授权的设备才能在下一个总线周期内使用总线。

PCI 总线没有规定总线仲裁的算法。仲裁器可以使用先到优先、循环优先级或其他算法。总线的授权只针对一次交换事务, 单次交换可以包含一个地址周期, 后面跟一个或多个数据周期。如果一个设备需要进行第二次数据交换, 而且这时没有其他设备发出总线请求, 它可以继续使用总线。即使如此, 在一般情况下, 两次交换之间需插入一个空闲周期。在特殊情况下, 即没有总线竞争情况, 设备可以不插入空闲周期而连续进行两次数据交换事务。如果总线设备正在进行一

次时间较长的数据传输，同时其他设备提出了总线申请，总线仲裁器就可以把 GNT#信号置反。当前使用总线的设备要监听 GNT#信号，当发现 GNT#被置反，它就必须在下个周期开始时释放总线的使用权，重新竞争。这样，在没有其他总线主设备提出总线请求时，可以使当前主设备进行长时间的数据传输，以达到高速数据传输，也可以在有竞争时对设备的请求进行快速响应。

5.3 直接程序传输方式与接口

主机和外部设备之间的 I/O 操作可采用 3 种常用的技术。

① 直接程序传输方式：数据在 CPU 和 I/O 设备间直接交换，CPU 执行程序来直接控制 I/O 操作，程序中由一系列的检测设备状态、发送读或写命令以及数据传输的指令组成。在这种方式中，当 CPU 发送一个命令后，它必须等待，直到外设操作完成。

② 程序中断方式：当 CPU 发送一个 I/O 指令后，继续执行其他操作，当外围设备完成工作后，再向 CPU 提请中断，请求对数据善后处理。

③ 存储器直接访问（Direct Memory Access, DMA）方式：在这种模式下，I/O 通过 DMA 接口直接实现和主存的交换数据，传输过程中不需 CPU 的参与。

直接程序传输方式是最简单的一种控制方式，是 CPU 直接利用 I/O 指令编程，实现数据的输入和输出。前两种方式中，无论是数据输入还是输出，都需要 CPU 直接负责数据的传输控制。

如果有关的操作时间固定且已知，可以直接执行输入指令或输出指令，如从某设备接口的缓冲区中读取数据，或向缓冲区输出数据。

如果有关操作时间未知或不定，如打印机的初始化操作或打印时的机电性操作，则往往采用查询、等待、再传输的方式。即在启动外部设备后，主机将不断通过 I/O 指令查询设备状态：是否准备好？或是否完成一次操作？若尚未准备好或尚未完成，CPU 将继续查询、等待；直到设备准备好，或完成一次操作，CPU 将通过 I/O 指令进行数据 I/O 传输。在外部设备工作期间，CPU 只执行与 I/O 有关的操作，即查询、等待、传输，这种方式又称为程序查询方式。

利用 I/O 指令实现数据 I/O 的相关接口，常见的有三种模式：一是具有中断功能的中断接口，而程序查询方式及不需查询的直接传输方式均可利用中断接口实现；二是按程序查询方式的需要设计接口；三是不需查询的简单接口。

接口的设计方案很多，一方面与 I/O 指令及系统总线有关，另一方面与外围设备有关。为了提供程序查询依据，接口中应设置状态字寄存器（或只有几位状态触发器），其中各位的设置方式也将影响到接口逻辑细节。图 5-13 给出了模型机的一种接口方案，以寄存器级功能模型粗框图描述，略去了门电路级细节。

接口中设置了数据缓冲寄存器与命令/状态寄存器，各占一个端口地址。当 CPU 访问本接口时，通过低 8 位地址总线送来地址码作为端口地址，再送来 IOR 或 IOW 命令，经译码后可选中某个寄存器，再通过数据总线进行数据输入/输出。

命令/状态寄存器的高位段作为命令字，可由 CPU 通过输出指令设置，体现 CPU 对外围设备的具体控制命令，经接口解释或直接送往设备。寄存器的低位段作为状态字，反映设备工作状态，提供给 CPU 作为程序查询依据，它也可由 CPU 初始化。现在状态字只设 2 位，分别为“忙”（Busy）

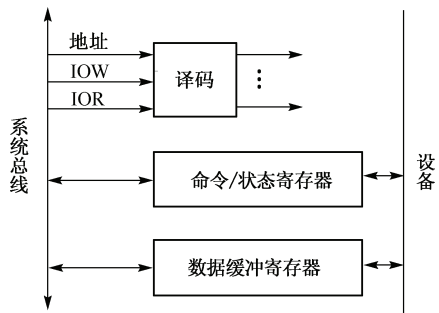


图 5-13 程序查询方式接口功能模型

和“完成”(Done)。

当不需要接口工作时,可通过复位命令,或由 CPU 编程设置,使 D (Done) 和 B (Busy) 均为 0,称为对接口清零。

如果需要启动外围设备工作, CPU 通过启动命令使 $D=0$ 与 $B=1$, 外围设备开始工作。此后 CPU 通过输入指令调入接口状态字,发现 $D=0$ 且 $B=1$, 知道外围设备还未准备好一次数据传输,就继续进行查询和等待。

如果是启动输入设备,则当数据送入接口的数据缓冲寄存器时,同时使 $D=1$ 且 $B=0$ 。CPU 通过读取状态字进行判别,得知接口已准备好输入数据,便执行输入指令,将数据经数据总线输入主机,并使 $D=0$ 且 $B=1$ 。

如果是启动输出设备,则当接口的数据缓冲寄存器有空时,使 $D=1$ 与 $B=0$ 。CPU 通过调入状态字判别,得知接口做好接收数据的准备,便执行输出指令,将数据经总线送至接口的数据缓冲寄存器,并使 $D=0$ 且 $B=1$ 。当接口将数据输出到外围设备后,数据缓冲寄存器再度“空”,又使 $D=1$ 且 $B=0$ 。

可见,程序查询方式体现的是这样一种编程策略:当 CPU 获知接口做好准备时,便执行 I/O 传输;当接口尚未准备好时, CPU 便等待并继续执行查询,而接口则为查询提供查询依据。如果 CPU 同时以程序查询方式启动多台 I/O 设备,则编程中可采取依次查询各设备的接口,并视情况做相应的处理。

模型机可以采用通用传输指令实现输入/输出操作,也可利用余下的操作码组合扩充显式 I/O 指令,相应地用端口地址选择接口寄存器。在这两种指令设置中, CPU 都需要先将状态字读入,再进行状态判别。有的计算机设置有专门的判转 I/O 指令,可直接根据接口的 D、B 状态实现判别与转移操作。

5.4 中断方式与接口

程序中断方式简称为中断,它几乎是所有计算机系统都应具备的一种重要机制,在实际工作中广泛应用。因此,许多课程都要从各自的角度阐述有关中断技术的知识,或者涉及与此有关的内容。前面章节着重从 CPU 角度介绍了基本概念,本节将进一步深入讨论中断方式,并从接口的角度介绍中断系统的组成及工作原理,这也是本章的重点内容。

5.4.1 中断的相关概念

1. 定义

程序中断方式是指:在计算机的运行过程中,如果发生某种随机事态, CPU 将暂停执行现行程序,转去执行中断处理程序,为该随机事态服务,并在服务完毕后自动恢复原程序的执行。由此可见,中断的这一定义包含了程序切换和随机性两个重要特征。

中断过程实质上是一种程序切换过程,由原来执行的程序切换到中断处理程序,处理完毕后再由中断处理程序切换为原来被暂停的程序。这就决定了程序中断方式的优点与不足,从而决定了中断方式的应用场合。由于是通过执行程序来进行对事件的服务处理,处理程序可以根据需要进行扩展,所以程序中断方式的处理能力很强,可以处理复杂事态。在实时控制系统中,许多实质性的功能模块就是以中断处理程序形态实现的,主控程序仅仅是一个组织各个模块的框架,程序量并不大。为了实现程序切换,决定了 CPU 在中断周期 IT 中的隐指令操作:保存断点、读取

服务程序入口地址；以及在转入服务程序后首先应执行的操作，如保护原程序的现场信息；在返回到原程序前，还需恢复现场、读取返回地址等。这一系列操作要花费一定的时间，使中断方式难以适应高速数据传输。因此，程序中断方式一般适用于处理中、低速的 I/O 操作和随机请求，所处理的对象可以是复杂的随机事态。

注意：常见的子程序调用和返回也存在程序切换，但这种切换是按严格约定进行的，并非响应随机事态。编程时在特定的代码位置有意调用子程序，还需为此约定参数。例如，调用一个浮点运算子程序，必须先准备好浮点数，并约定这些操作数所在的存储位置，子程序本身也需约定运算结果的保存位置。显然，子程序的调用不能随机插入到主程序之中。

中断与转子程序两者的本质区别主要表现在：

① 转子子程序的执行是由程序员事先安排好的，中断服务程序的执行则是由随机的中断事件引起的调用。

② 转子子程序的执行受到主程序或上层子程序的控制，中断服务程序一般与被中断的现行程序没有关系。

③ 不存在一个程序同时调用多个转子子程序的情况，却经常可能发生多个外设同时请求 CPU 为自己服务的情况。

与一般的转子含义不同，程序中断方式的主要特点是具有随机性。初学者不难理解这一点，但遇到实际问题时，往往不知道如何对随机事态安排程序中断，以及为随机出现的事件编制中断服务程序。为了深入理解这一特点，我们将中断方式的随机性进一步分为：随机出现的事件；主机在宏观上有意调用外围设备，但以随机请求提出方式实现服务处理；随机插入的软中断等几类。下面结合中断方式的典型应用加以说明。

2. 中断方式的典型应用

（1）以中断方式管理中低速 I/O 操作，使 CPU 与外部设备并行工作

像键盘一类的设备，工作时主动地向主机提出随机请求。我们编程时并不能确切地知道何时按键，如果让 CPU 以程序查询方式管理键盘，CPU 将不能执行自身的处理任务。所以，需要以中断方式管理键盘。平时 CPU 执行其程序，当按下某个键时，键盘产生中断请求，CPU 转入键盘中断服务程序，获取按键编码，并根据键码要求做相应处理。这种情况被称为随机出现的事件。

像打印机一类的设备是由 CPU 在程序的特定位置有意调用的。如果采用中断方式管理，在启动打印机后，CPU 仍可继续执行原先安排的程序，因为打印机启动后还需要一段初始化准备过程。当打印机做好准备可以接收打印信息时，将提出中断请求。CPU 转入打印机中断服务程序，将一行打印信息送往打印机，然后恢复执行原程序，此时打印机进行打印。当打印完一行后，打印机再次提出中断请求，CPU 再度转入中断服务程序，送出又一行打印信息。如此循环，直至全部打印完毕。这种管理方式被称为“有意调用，随机处理”。相应地，在编程时采取这样一种方式：在准备好打印的信息（一般是送入主存的一个输出缓冲区）后，启动打印机，然后编写可并行执行的程序；将打印机中断服务程序作为一个独立模块，单独编写，以便响应中断时调用。由于打印机是一种机电型设备，其打印时间较长，如果有事可干，CPU 利用打印机做初始准备与打印的时间，并行地执行自身的程序。

虽然中断方式一般只适于管理中、低速 I/O 操作，但磁盘一类高速外设中也包含中低速的机电型操作，如磁盘寻道等，所以磁盘接口一方面按 DMA 方式实现数据传输，同时具备中断功能，用于寻道判别和结束处理等。

（2）软中断

许多计算机系统都设置有软中断指令，如指令“INT n”，这里的n为中断号。有的计算机将它称为程序自愿中断。执行“INT n”指令，将以响应随机中断请求方式进行服务程序处理，并切换到服务程序。

初看起来，执行软中断指令与执行转子指令似乎相似，其实它们在处理方法上是有区别的。如前所述，转子指令（子程序调用）只能按严格的约定，在特定位置执行。而软中断指令的执行是作为中断来处理的，在中断周期中保存断点，按软中断指令给出的中断号来查找中断向量表，找到相应的中断服务程序入口，实现程序切换。因此，软中断可以随机插入程序的任何位置，我们将它的随机性理解为“有意调用，随机插入”。

早期，软中断用于设置程序断点，引出调试跟踪程序，分析原程序执行结果，帮助调试。现在操作系统常为用户提供一种操作界面，称为系统功能调用，即由系统软件编制者将用户常用的一些系统功能（如打开文件、复制文件、显示、打印、跟踪调试程序等）事先编成若干中断服务程序模块，纳入操作系统的扩展部分。用户通过执行软中断指令，调用所编制的中断程序。虽然在有些操作系统中，这种系统功能调用实际上是按一般子程序调用方式编写，不能随机插入，但仍利用响应中断的方式与机制实现。

从程序模块之间的关系看，中断服务程序是临时嵌入的一段，所以又称为中断处理子程序。原程序被打断，以后又自动恢复，作为程序的主体，常将它称为主程序。这是广义上的主程序和子程序，与指令系统中的转子和返回过程是有区别的。

（3）故障处理

计算机工作时可能产生故障，但何时出现故障？是什么故障？显然是随机的，只能以中断方式处理。即事先估计到有可能出现哪些故障，如果出现这些故障应当如何处理，编成若干故障处理程序模块；一旦发生故障，提出中断请求，转故障处理程序进行处理。

常见的硬件故障有掉电、校验错、运算出错等。大多数计算机都有掉电处理和校验错处理功能。当电源检测电路发现电压不足或掉电时，提出中断请求，利用直流稳压电源滤波电容的短暂维持能力（毫秒级），进行必要的紧急处理，如将关键信息存入由后备电池供电的CMOS存储器中，或将电源系统切换为UPS电源。当产生校验出错时，提出中断请求，一般处理方法是重复读出，判断是否偶然性故障；如果是永久性故障，将显示出错信息，操作员可以考虑停止运行。有些计算机具有运算出错的判断能力，从而也可提出中断请求。

常见的软件故障如溢出、地址越界、使用非法指令等。在定点运算中，由于比例因子选择不当可能产生溢出，通过溢出判别逻辑可引发中断，在中断处理中修改比例因子，重新启动有关运算过程。在多道程序工作方式中，操作系统为各用户分配了存储空间，如果某用户程序访存地址越界，则可由地址检查逻辑引发中断，提示用户修改。许多计算机将执行程序状态分为用户态和管态，用户态执行用户程序，管态执行系统管理程序。有少数指令是为编制系统管理程序专门设置的，称为特权指令。如果用户程序误用这些特权指令，称为非法指令，将引发故障中断。

（4）实时处理

实时处理是指在事件出现的实际时间内及时地进行处理，而不是积压起来留待以后批量处理。实时程度视具体应用需要而定。这是计算机的一个重要应用领域，如巡回检测系统、各种生产过程的计算机控制系统等，都属于实时处理系统。广泛应用中断技术，中断服务程序量可能很大，甚至占应用程序的大部分。

在实时控制系统中，常设置实时时钟，定时地发出实时时钟中断请求。CPU 转入中断服务程序，在其中采集有关参数，与要求的标准值进行比较，当有误差时按一定控制算法进行实时调整，以保证生产过程按设定的标准流程，或按优化的流程进行。

如果需要实时监控的对象发生异常，也可直接提出中断请求，以便及时处理。

(5) 多机通信

在多机系统和计算机网络中，各节点之间需要相互通信，以便交换信息或协同工作。当一个节点要与其他节点通信时，便向目标节点提出中断请求，对方接受请求后就转入中断服务程序，以实现相互通信。

(6) 人机对话

现代计算机越来越强调良好的人机交互界面。用户可以通过键盘终端或其他设备向计算机输入命令和数据，主机则通过显示器输出设备，提供运行结果和有关状态。或者，用户向计算机提出某种询问，计算机给出提示、回答。在信息检索系统中，显示器常以菜单形式供操作者做出选择；或者由计算机主动显示执行情况，给操作者提供干预的可能等。这种交互式操作被形象地称为人机对话，其操作也带有明显的随机性（何时提出询问或要求？何时做出回答？），因而也以中断方式提出与处理。

可见，中断方式不仅用于 I/O 操作的控制，它的应用极为广泛。我们没有必要也不可能穷举所有应用实例，希望读者从上述例子中得到启示，能够举一反三。

3. 中断系统的硬件、软件组织

常将与中断功能有关的硬件、软件统称为中断系统。第 3 章中介绍了 CPU 中的有关中断批准逻辑、中断周期中的隐指令操作等部分。有关接口方面的硬件组成原理将在后面几节中深入讨论。现在先介绍有关的软件组织。

由于中断请求出现的随机性，无法在主程序的预定位置进行处理，需要独立地编制中断处理程序。现以模型机为例，提供一种比较典型的软件组织方法。

(1) 列出系统需要处理的各种中断请求

模型机外部硬件中断源：IRQ₀ —系统时钟，如日历钟；IRQ₁ —实时时钟，供实时处理用；IRQ₂ —通信中断，组成多机系统或联网时用；IRQ₃ —键盘；IRQ₄ —CRT 显示器；IRQ₅ —硬盘；IRQ₆ —软盘；IRQ₇ —打印机。

如果实时处理需要的中断源较多，可通过 IRQ₁ 和 IRQ₂ 进行扩展。

模型机内部硬件中断源有掉电中断、溢出中断、校验错中断。

模型机软中断有 INT 11~INT n，可以根据需要进行扩充，作为系统的功能调用命令。

(2) 针对各中断源的需要，分别编制对应的中断服务程序模块

这些服务程序在主存中的存储空间不必连续，允许分散存放和补充。

(3) 将各中断服务程序的入口地址写入中断向量表

在模型机 CPU 设计时，我们将主存的 0 号和 1 号单元用于复位时转入监控程序入口，所以中断向量表从 2 号单元开始。中断向量表中存放着各中断服务程序的入口地址（未考虑中断服务程序状态字），称为中断向量。模型机只用 16 位地址，并按字编址，所以每个中断向量占一个编址单元。访问中断向量表的地址称为向量地址，在模型机中，向量地址=中断号+2。例如，IRQ₀ 所对应的服务程序入口地址，存放在 0+2=2 号单元；INT 11H 所对应的服务程序入口地址，则存放在 13 号单元之中，其他软中断指令可以以此类推。

按照这样的软件组织方法，中断服务程序是独立于主程序事先编制的。在编制用户主程序时只须提供允许中断的可能（如开中断），不必细致考虑何时中断以及如何处理中断等，也不必考虑中断服务程序如何嵌入主程序。一旦发生中断请求，可通过硬件中断请求信号或软中断指令“INT n”提供的中断号，经过一系列转换得到向量地址，据此从向量表中找到相应的服务程序入口地址，从而转入中断服务程序执行。

4. 中断的分类

（1）硬件中断和软件中断

硬件中断指由某个硬件中断请求信号引发的中断，软中断是由执行软中断指令（软件）所引起的中断。处理上几乎相同，其区别仅在于：硬件中断通过中断请求信号形成向量地址，而软中断由指令提供中断号，再被转换为向量地址，后续的响应和处理过程几乎相同。

（2）强迫中断和自愿中断

强迫中断是由于故障、外部请求等所引起的强迫性中断，非程序本身安排的，这种请求的提出和相应的服务处理都是随机的。

自愿中断又称为程序自陷中断，即软中断。这是程序有意安排的，以中断方式引出服务程序，实现某种功能。这种中断虽是有意安排的，但可以随机插入。

（3）内中断和外中断

内中断指来自主机内部的中断请求，如掉电中断、CPU 故障中断、软中断等。外中断指中断源来自主机外部，一般指外部设备中断，如时钟中断、键盘中断、显示器中断、打印机中断、磁盘中断等。本章从输入/输出子系统角度，重点讨论外中断的提出、传递、排优、响应、处理及相应的中断接口模型。

（4）可屏蔽中断和非屏蔽中断

外中断请求是由于某个外部设备（接口）或某个外部事件的需要而提出的，但 CPU 可对此施加某种控制，其中一项基本方法就是屏蔽技术。CPU 可向外围接口送出屏蔽字代码，每位可屏蔽一种中断源，不允许它提出中断请求，或者不允许它已经发出的中断请求信号送达 CPU，因此可将中断分为可屏蔽中断和非屏蔽中断两种。有些微处理器芯片，如 Intel 8086/8088，将其中断请求输入端分为可屏蔽中断 INTR 和非屏蔽中断 NMI 两种。

一般的外围设备中断都是可屏蔽的中断，CPU 通过屏蔽技术施加控制。一些必须响应的中断请求（如掉电、故障等引起的中断），作为非屏蔽中断，不受 CPU 屏蔽。此外，软中断发生于 CPU 内部，不属于外中断范畴，从概念上讲，它也是不可屏蔽的。

（5）向量中断和非向量中断

如何形成中断服务程序的入口地址，这是向量中断和非向量中断最本质差别。

如果直接依靠硬件，通过查询中断向量表来确定入口地址，就是向量中断方式；如果是通过执行软件（如中断服务总程序）来确定中断服务程序的入口地址，则属于非向量中断。

向量中断和非向量中断的有关概念将在 5.4.4 节中深入讨论。计算机一般具有向量中断功能，但可运用非向量中断思想对向量中断方式进行扩展。前面介绍的模型机中断系统将各中断服务程序的入口地址组织成一个中断向量表，这种方式就属于向量中断。

在介绍了上述一些必要的基本概念之后，下面将沿着外中断的全过程，深入地分析中断系统的硬件组成及工作原理。

5.4.2 中断请求

1. 中断请求逻辑

一个中断的出现会触发一系列的事件发生，而要形成一个设备的中断请求逻辑，则需同时具备以下逻辑条件。

① 外部设备有中断请求的需要，如“准备就绪”或“完成一次操作”，可以用完成触发器状态 $T_D=1$ 表示。例如，打印机接口可接收打印数据时， $T_D=1$ ；键盘接口在可输出键码时， $T_D=1$ 。

② CPU 没有对该中断源屏蔽，允许提出请求，可以用屏蔽触发器状态 $T_M=0$ 表示。相应地，可将接口中与中断有关的逻辑设置为两级。一级是反映外部设备与接口工作状态的状态触发器，如以前讨论过的忙触发器 T_B 和完成触发器 T_D ，它们是组成状态字的有关位，反映了提出请求的需要。另一级是中断请求触发器 IRQ ，表明是否形成中断请求。至于中断屏蔽，可采取分散屏蔽或集中屏蔽。

分散屏蔽是指 CPU 将屏蔽字代码按位分送给各中断源接口，接口中各设一位屏蔽触发器 T_M ，接收屏蔽字中对应位的信息，为 1 则屏蔽该中断源，为 0 则不屏蔽。

一种方法是在中断请求触发器 IRQ 的 D 端进行屏蔽，如图 5-14(a)所示。如果 $T_D=1$ 、 $T_M=0$ ，则同步脉冲将 1 送入请求触发器，发出中断请求信号 IRQ 。另一种方法是在中断请求触发器的输出端进行屏蔽，如图 5-14(b)所示。

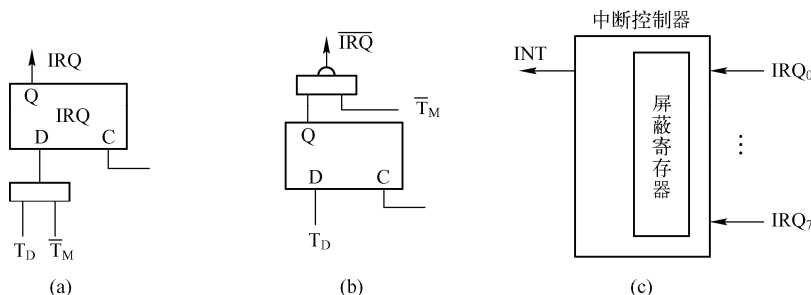


图 5-14 中断请求逻辑

图 5-14(c)是采取集中屏蔽方法，即在公共接口逻辑中设置一个中断控制器（有集成芯片如 Intel 8259），内含一个屏蔽字寄存器，CPU 将屏蔽字送入其中。对各中断源的接口不另设屏蔽触发器，一旦 $T_D=1$ ，即可提出中断请求信号 IRQ 。将各请求信号汇集到中断控制器，并与屏蔽字比较。若未被屏蔽，则中断控制器送出一个公共的中断请求信号 INT ，此请求信号再送往 CPU。

中断请求的提出可以采用同步定时，见图 5-14(a)、(b)，同步脉冲 CP 加到中断请求触发器 IRQ 的 C 端，也可以不采用同步定时，具备请求条件（如 $T_D=1$ ）时由 S 端置入 IRQ ，立即发出中断请求信号。CPU 响应中断请求最后采取同步控制方式。

2. 中断请求信号的传输

一台计算机系统中有多个中断源，可能产生多个中断请求信号，它们是如何传输给 CPU 的呢？一般有 4 种中断请求信号传输模式，如图 5-15 所示。

① 各中断源单独设置自己的中断请求线，多根请求线直接送往 CPU，如图 5-15(a)所示。当 CPU 接到中断请求信号后，立即知道请求源是哪个设备，这有利于实现向量中断，因为可以通过编码电路形成向量地址。但 CPU 所能连接的中断请求线数目有限，特别是微处理器芯片引脚数有限，不可能给中断请求信号分配多个引脚，因此中断源数目难以扩充。

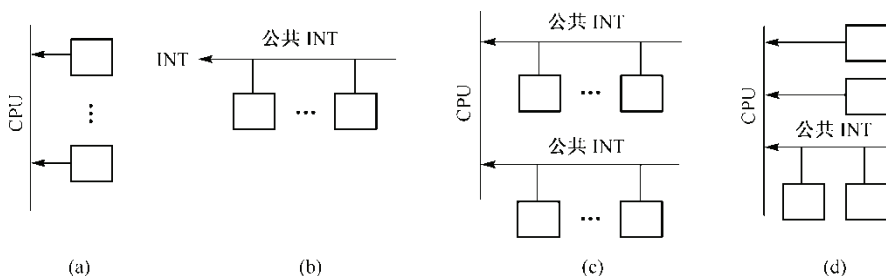


图 5-15 中断请求信号的传输模式

② 各中断源的请求信号通过三态门汇集到一根公共请求线，如图 5-15(b)所示。只要负载能力允许，挂在公共请求线上的请求源可以任意扩充，对于 CPU 来说，只需接收一根中断请求线即可。这种连接逻辑也可在中断控制器芯片 8259 内部实现，如图 5-14(c)所示，多根请求线 IRQ_i 输入 8259，在芯片内汇集为一根公共请求线 INT 输出。采用这种方式，CPU 需要通过一定逻辑来识别被批准的中断源，也可在 8259 芯片内实现。微型计算机中广泛采用这种连接模式。

③ 一种折中方案是采用二维结构，如图 5-15(c)所示。CPU 设置数根中断请求输入线，它们体现不同的优先级别，称为主优先级。再将主优先级相同的中断请求源汇集到该公共请求线上。这就综合了前两种模式的优点，既可以在主优先级层次迅速判明中断源，又能随意扩充中断源数目。在小型计算机中，允许 CPU 对外连线数目超过一般的微型计算机 CPU 芯片所能支持的最大对外连线数，因此常选用这种连接模式。

④ 另一种折中方案是兼有公共请求线与独立请求线，如图 5-15(d)所示。将要求快速响应的 1~2 个中断请求采取独立请求线方式，以便快速响应中断源的请求。再将其余响应速度允许相对较低一些的中断请求，汇集为一根公共请求线。有些微处理器由于引脚数有限，因此常采取这种信号线的混合连接模式。

5.4.3 中断判优

当外部提出中断请求时，CPU 是否响应？这取决于 CPU 现程序的优先级高还是中断请求的优先级更高。当有两个以上的中断源同时提出请求时，CPU 首先响应谁的请求？这就要求中断系统具有相应的优先级判断逻辑和优先级动态调整的手段。

1. CPU 与中断请求之间的判优

CPU 的状态就是现程序的执行状态，外中断请求则是外部设备的请求，广义一点说，是外部事件的服务请求。一般有两种手段可用于处理它们之间的优先级问题。

首先，几乎所有 CPU 都设置了一个“允许中断”触发器 T_{IEN} ，指令系统提供开中断与关中断功能，开中断操作使 $T_{IEN}=1$ ，关中断使 $T_{IEN}=0$ 。如果关中断，则不响应外中断请求。换句话说，此时所有外中断请求的服务都没有现程序的任务重要。如果开中断，则可响应外部请求。在一般微型计算机中只有这一级的中断控制。

性能更强的计算机，除了设置 T_{IEN} 触发器来开、关中断外，还可在程序状态字 PSW 中设定现程序的优先级，以标志现程序的重要程度。CPU 有一个优先级比较逻辑，对 PSW 中给定的优先级与中断请求的优先级进行比较，决定是否需要暂停现程序去响应中断请求。用户编程时，如果认为某段程序任务比较紧迫，希望不被打断，则可在该程序段的起始处，将 PWS 中的优先级设定得高些。当然，不恰当地提高现程序的优先级会降低 CPU 对外部事件的响应能力。

反之，如果某段程序任务相对不很紧迫，允许暂时搁置，或者在某个时刻需要处理 I/O 操作或其他外部事件，则可先将程序优先级降低。中断请求线一旦连好，则其优先级基本确定，CPU 现行程序优先级则可根据需要动态修改。

2. 中断请求之间的判优

在各种请求之间，根据什么原则安排优先级别呢？按请求的性质，一般的优先顺序是：故障引发的中断请求 → DMA 请求 → 外部设备中断请求。这样安排是因为处理故障的紧迫性最高，DMA 请求是要求高速数据传输，高速操作一般应比低速操作优先。

按中断请求要求的数据传输方向，一般原则是输入操作的优先级高于输出操作。如果不及时响应输入操作请求，有可能丢失输入信息。而输出信息一般存于主存或者缓存中，暂时延缓一些，也不至于丢失信息。

当然，上述原则也不是绝对的，在设计时必须具体分析。在多数计算机中，一方面，用硬件逻辑实现优先级判别，常简称为排优逻辑，即按优先级排队。在硬件判优逻辑中，各中断源的优先级是固定的。另一方面，计算机又可改用软件查询方式体现优先级判别，则可动态地调整优先级。此外，采用屏蔽技术也可在一定程度上动态调整优先顺序。下面介绍几种优先级排队方法。

(1) 软件查询

响应中断请求后，先转入查询程序，按优先顺序依次询问各中断源是否提出请求。如果查询到中断源，则转入相应的服务处理程序；如果没有，则继续往下查询。查询的顺序体现了优先级别的高低，先被询问则优先级更高，改变查询顺序也就改变了优先级。

如前所述，有些计算机设置了专门的查询 I/O 指令，可以直接根据外围接口中状态触发器的状态，进行判别与转移；也可以用输入指令或通用的传输指令取回状态字，进行判别；或者在公共接口中设置一个中断请求寄存器，用来存放各中断源提出的请求信息（对应位为 1，表示该中断源提出了请求）。在进行软件查询时，就可将中断请求寄存器的内容取回 CPU，按照优先顺序逐位判定。

采用软件查询方式进行判优，不需要硬件判优逻辑，可以根据需要灵活地修改各中断源的优先级；但通过程序逐个查询，所需时间较长，特别是对优先级较低的中断源不太公平，可能需要查询多次后才能轮上。因此，软件查询方式通常只适宜应用在低速的小型系统中，更多时候，它是作为硬件判优逻辑的一种软件补充手段。

(2) 并行优先级排队逻辑

如果各中断源都能提供独立的中断请求线连接到 CPU，则可以采取并行优先级排队逻辑，即具有独立请求线的硬件优先排队逻辑，如图 5-16 所示。

各中断源的中断请求触发器向排优逻辑电路送出自己的请求信号： $INTR'_0$ 、 $INTR'_1$ 等。经过排优电路，向 CPU 送出中断请求信号 $INTR_0$ 、 $INTR_1$ 等。

这种排优逻辑的工作原理一目了然： $INTR'_0$ 的优先权最高， $INTR'_1$ 次之，以此类推。如果优先权高的中断源 i 提出了中断请求 $INTR'_i$ ，则它自动封锁比它优先级更低的所有其他请求。仅当优先者没有请求时，才允许次一级的中断请求有效。因此，如果同时有几个 $INTR'_i$ 提出，则只有其中优先级最高者能向 CPU 送出 $INTR_i$ 有效信号，其余请求信号均被封锁。在这种排优逻辑中，判优的最终结果表现为中断源的请求信号是否有效，即是否允许发出请求信号 $INTR_i$ 。

上述并行排优逻辑适于具有多请求线的计算机系统，响应速度较快，但硬件代价较高。

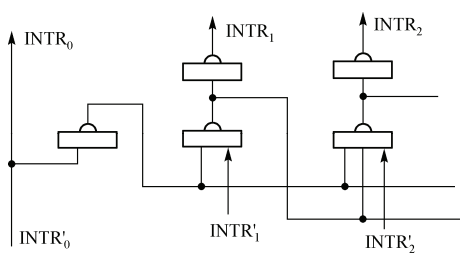


图 5-16 具有独立请求线的并行排优逻辑

(3) 链式优先排队逻辑

如果中断请求信号的传递模式采用公共请求线，即汇集为一个公共的请求信号，则判优结果可用不同的设备码或用中断源类型码来表示，相应地可采用链式排优逻辑结构，如图 5-17 所示，称为优先链。各中断源提出的请求都送到公共请求线上，形成公用的中断请求信号 INT，送往 CPU。响应请求时，CPU 向接口发出一个公用的批准信号 INTA。

在图 5-17(a)结构中，批准信号同时送往所有接口，但优先链使得只有在申请者中优先级最高者，可以通过系统总线向 CPU 送出自己的编码。这种编码可以是被批准者的设备码，或是该中断源的类型码（中断号），或是直接形成访问中断向量的向量地址码。根据编码，CPU 可以识别所批准的中断源，从而转向对应的中断服务程序。限于篇幅，对具体的编码电路、控制发送编码的优先链电路略去未画。批准信号 INTA 起到查询中断源的作用，它是同时向所有中断源发出的，所以称为多重查询方式。

在图 5-17(b)结构中，CPU 发出批准信号 INTA，首先送给优先级最高的设备。如果该设备提出了请求，则在接到批准信号后，通过系统总线向 CPU 送出自己的编码（设备码，或中断类型码，或向量地址码），批准信号的传输也就到此为止，不再往下传输。如果该设备没有提出请求，则将批准信号传向下一级设备，看其是否提出请求……采取这种连接方法，使所有可能作为中断源的设备连接成一条链，连接顺序体现了优先顺序。在逻辑上离 CPU 最近的设备，优先级最高。这种优先链结构在许多文献中被称为菊花链，这是应用得很广泛的一种逻辑结构（具体的逻辑门级电路略去未画）。

(4) 二维结构的优先排队

如果中断请求信号的传输采取二维结构，则优先排队逻辑结构如图 5-18 所示。CPU 可以接受 $n+1$ 根中断请求线，它们的优先级称为主优先级，在 CPU 内部有一个相应的判优电路，用以首先响应优先级最高的请求。如果程序状态字中有 CPU 现程序的优先级编码，这个判优电路同时担负 CPU 与请求之间的判优任务。通常，将 DMA 请求也纳入这个二维优先结构之中，占有主优先级中最高一级。

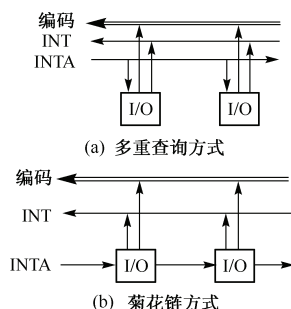


图 5-17 优先链排队逻辑

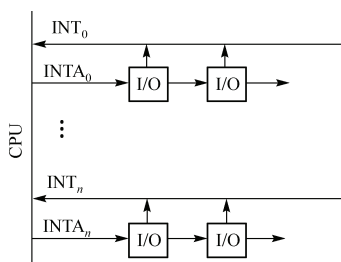


图 5-18 二维结构的优先排队逻辑

将有关外部设备分成 $n+1$ 组，每组的请求汇集到同一根请求线上，占有同一个主优先级。在一个小组内，各设备又进一步进行优先级划分，称为次优先级。通常在组内采取菊花链式的优先链结构。

(5) 采用中断控制器集成芯片的优先逻辑

在微型计算机中，广泛使用一种中断控制器集成芯片，如 Intel 8259A 将中断请求信号的寄存、汇集、屏蔽、排优、编码等逻辑，集成在一块芯片之中。在设计中断系统时，使用这种芯片就非常方便，不必了解芯片内究竟使用何种具体的排优逻辑，如图 5-19 所示。

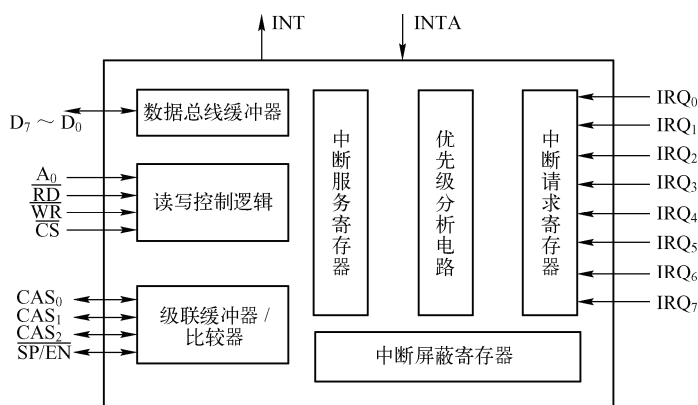


图 5-19 中断控制器 8259A 的内部结构

8259A 芯片中包含下述几个重要的组成部分。

① 中断请求寄存器（Interrupt Request Register, IRR）：8 位，可存放 8 个中断请求信号，作为向 CPU 申请与判优、编码的依据。

② 优先级裁决器（Priority Resolver, PR）：即中断源的优先级排队逻辑，选择优先级最高的中断申请者。

③ 中断服务寄存器（Interrupt Service Register, ISR）：8 位，用来存放或记录正在服务中的所有中断请求优先级（如在多重嵌套时），每位对应一个中断优先级。

④ 中断屏蔽寄存器（Interrupt Mask Register, IMR）：8 位，其内容可由 CPU 预置。这就是前面提到的集中屏蔽方式，各接口可以提出自己的中断请求信号，在 8259A 中再与屏蔽字比较。对应位（ IMR_i ）若为 1，则该请求（ IRQ_i ）被屏蔽，不会被送往 CPU。

⑤ 数据总线缓冲器：一个三态 8 位数据缓冲器，8 位数据线 $D_0 \sim D_7$ 与系统的数据总线相连，用来进行 CPU 与 8259A 之间的数据传输，当 CPU 从 8259A 进行读操作时，数据总线缓冲器用来传输从 8259A 内部送往 CPU 的数据/状态信息和中断类型码；写操作时由 CPU 向 8259A 内部传输数据/控制字。

⑥ 读写控制逻辑：该部件接收来自 CPU 的读写命令，由 CS、RD、WR 和地址线 A_0 共同控制，完成规定的操作。其中 A_0 引脚功能一般直接与 CPU 地址总线的 A_0 位相连， A_0 为 0 或 1 时，可以选择芯片内不同的寄存器进行读或写。

⑦ 级联缓冲器/比较器：提供级联控制信号 $CAS_0 \sim CAS_2$ 及双向功能信号 SP/EN，以满足 8259A 在缓冲工作和主从工作（即非缓冲工作）两种方式下的功能需要。

⑧ 控制逻辑：8259A 全部功能的核心，包括一组方式控制字寄存器和一组操作命令字寄存器以及相关的控制电路。芯片的全部工作过程完全由上述两组寄存器内容设定，这两组寄存器可以通过编程写入不同的参数，进行预先设置。

在输入/输出子系统中，中断控制器 8259A 作为公共接口逻辑，一般位于主机板上。它接收各路中断请求信号 $IRQ_0 \sim IRQ_7$ ，将它们存放于中断请求寄存器中。并将未被屏蔽的中断请求送入优先级分析电路参加判优，产生一个公共的中断请求信号 INT，送往 CPU。

CPU 响应中断请求时，发出批准信号 INTA，送往 8259A。然后 8259A 设置 ISR 相应位为 1 并复位 IRR 相应位为 0，表明此中断请求正在被 CPU 处理，而不是正在等待 CPU 处理。

随后，CPU 会再次发送一个 INTA 信号给 8259A，以询问中断源，8259A 根据被设置的起始向量号（由控制字初始化）与当前中断请求号计算出该请求的中断向量号（也称中断类型码），

并将其通过 $D_0 \sim D_7$ 向数据总线输出。CPU 从总线上接收到该中断向量号，以此为依据查找中断向量表，获取中断服务程序入口地址，再转向对应的中断服务程序执行。

8259A 中断控制器芯片也可以多级串联(级联)，将一片 8259A 的 INT 输出作为上一级的 IRQ_i 输入，以扩展系统可支持的中断源数量。

3. 屏蔽技术的应用

前面已经提到了中断屏蔽技术，它的基本含义是通过输出指令送出一个屏蔽字，有选择地允许某些中断请求，屏蔽某些中断请求，这一技术常用于如下两种场合。

(1) 在多重中断方式中实现中断嵌套

当 CPU 响应某个中断请求后，送出一个新的屏蔽字，以禁止与该请求同一优先级或更低优先级的其他请求。只允许提出比该请求优先级高的其他中断请求，使 CPU 有可能暂停现行中断服务程序，转去执行更紧迫的中断处理任务；而低于或等同于该请求级别的请求则被屏蔽，不对现行中断处理任务造成干扰。

(2) 利用屏蔽技术动态地修改优先级

利用硬件排优逻辑所分配的优先级是固定的，但有时需要动态地修改优先顺序。例如，有些设备的优先级低，经常得不到响应的机会，在适当的时候需要修改设备的优先级，使各设备得到均衡、合理的响应机会。因此，可在一段时间内，利用屏蔽字将原来优先级高的设备请求暂时屏蔽。原来级别低的请求由于未被屏蔽，优先级相对提高，称为中断升级。过一段时间还可以再进行屏蔽字调整，或者复原最初屏蔽字，或者按一定规律不断地修改屏蔽字，以动态地适应程序的需要。

以上围绕中断优先级问题讨论了多种方法，有些方法可以综合运用，从而在实际应用中还可以派生出许多具体方式。例如，中断控制器 8259A 可编程指定多种工作方式，如固定优先级、循环优先级、特殊屏蔽方式等，有关细节在后续课程中再详细介绍。

5.4.4 中断响应

CPU 响应中断后，通过执行中断服务程序进行中断处理。服务程序事先存放在主存中，为了转向中断服务程序，关键是获得该服务程序的入口地址。因此我们先讨论如何获得服务程序的入口地址，再说明响应中断的条件和中断响应过程。

1. 中断服务程序入口地址的获取方式

如前所述，可通过向量中断方式（硬件方式）或非向量中断方式（软件查询方式）获取中断服务程序的入口。

(1) 向量中断方式

首先，我们定义 3 个有关的概念。

① 中断向量。采用向量化的中断响应方式，将中断服务程序的入口地址及其程序状态字存放在特定的存储区中，所有的中断服务程序入口地址和状态字一起，称为中断向量。有些计算机（如普通的微型计算机）没有完整的程序状态字，则中断向量仅指中断服务程序的入口地址。一个计算机系统需为若干中断源编制相应的服务程序模块，各有其入口地址。每个入口地址是一个标量（或视为布尔量），这些入口地址被组织为标量的一维的有序集合，因而可以视为向量，即为中断服务的向量，简称中断向量。

② 中断向量表，即用来存放中断向量的一种表。在实际的系统中，常将所有中断服务程序

的入口地址（或包括服务程序状态字）组织成一个一维表格，并存放于一段连续的存储区，此表就是中断向量表。

③ 向量地址，访问中断向量表的地址码，即读取中断向量所需的地址（也可称为中断指针）。

向量中断是指这样一种中断响应方式：将各个中断服务程序的入口地址（或包括状态字）组织成中断向量表；响应中断时，由硬件直接产生对应于中断源的向量地址；按该地址访问中断向量表，从中读取服务程序入口地址，由此转向服务程序。这些工作一般在中断周期中由硬件直接实现。

向量中断的特点是根据中断请求信号快速地直接转向对应的中断服务程序。因此，现代计算机基本上都具有向量中断功能，其具体实现方法可以有多种。

在 IBM PC 系统中，中断向量表存放在主存的 0~1023（十进制）单元中，如图 5-20 所示。每个中断源占用 4 字节，存放其服务程序入口地址，其中 2 字节存放其段地址，2 字节存放偏移量。因此，整个中断向量表可以驱动 256 个中断源，与中断类型码 0~255 相对应。对于 8086/8088 CPU 的系统，中断向量表分为三部分：第一部分是专用区域，对应于中断类型码 0~4，为系统定义的一些内部中断源和非屏蔽中断源；第二部分是系统保留区，中断类型码 5~31，为系统的管理调用和留作新功能的发展；第三部分是留给用户使用的区域，中断类型码 32~255。

当 CPU 响应中断请求时，向 8259A 中断控制器送去批准信号 INTA；并由数据总线从 8259A 取回被批准请求源的中断类型码；乘以 4，形成向量地址；访问主存，从中断向量表中读得服务程序入口地址；然后转向服务程序。如果中断类型码为 0，则从 0 号单元开始，连续读取 4 字节的入口地址（段基址及偏移量）。如果中断类型码为 1，则从 4 号单元至 7 号单元读取其入口地址。

当 CPU 执行软中断指令“INT n”时，将中断号 n 乘以 4，形成向量地址；访问主存，从中断向量表中读取服务程序入口地址。可见，软中断是由软中断指令给出中断号即中断类型码 n，而外部中断是由某个中断请求信号 IRQ_i 引起的，经中断控制器转换为中断类型码 n。

在 80386/80486 系统中，性能较之 8086/8088 有所提升。中断向量表可以存放在主存的任何位置，将向量表的起始地址存入一个向量表基址寄存器中。中断类型码经转换后，形成距向量表基址的偏移量。80386/80486 的访存有实地址方式与虚拟地址保护方式之分。在实地址方式中，物理地址 32 位，每个中断源的服务程序入口地址在中断向量表中占 4 字节（与 8086/8088 系统相似）。在虚地址方式中，虚地址 48 位，每个中断源在中断向量表中占用 8 字节，其中 6 字节给出 48 位以虚地址编址的服务程序入口地址，其余 2 字节存放状态信息。

除上述两种外，产生向量地址的方法还有多种。如在具有多根请求线的系统中，可由请求线编码产生各中断源的向量地址。又如，在菊花链结构中，经由硬件链式查询找到被批准的中断源，该中断源通过总线向 CPU 送出其向量地址。再如，中断源送出一种复位指令 RST n 代码，再转换成向量地址。在 IBM PC 和 80386/80486 中，中断源产生的是偏移量，与 CPU 提供的中断向量表基址相加，形成向量地址。在有些系统中，CPU 内有一个中断向量寄存器，存放向量地址的高位部分，中断源产生向量地址的低位部分，二者拼接形成完整的向量地址。

另一种使用向量中断的技术是总线仲裁。对于总线仲裁，I/O 模块在引发中断请求线前必须首先获得总线控制权，因此一次只有一个模块引发这条线。当 CPU 检测到中断时，在中断响应线

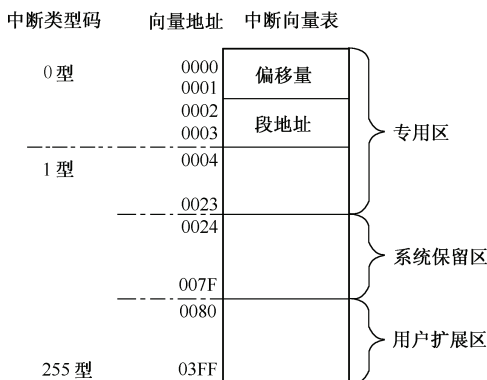


图 5-20 IBM PC 的中断向量表

上响应，然后请求中断的外设把它的向量放在数据线上。

(2) 非向量中断方式

非向量中断是指这样一种中断响应方式：CPU 响应中断时只产生一个固定的地址，由此读取中断查询程序（也称为中断服务总程序）的入口地址，然后转向查询程序并执行；通过软件查询方式，确定被优先批准的中断源，然后获取与之对应的中断服务程序入口地址，分支进入相应的中断服务程序。

例如在 DJS-130 机中，CPU 响应中断时，在中断周期中让 PC 和 MAR 的内容均为 1，从 1 号存储单元中读出查询程序的入口地址；然后转向查询程序，通过执行查询程序，按优先顺序逐个地查询各中断源。若某中断源提出了中断请求，则转向相应的服务程序；若未提出请求，则继续往下询问。

查询程序是为所有中断请求服务的，因此又称为中断总服务程序。它的任务仅仅是判定优先的、提出请求的中断源，从而转向实质性处理的服务程序。查询程序本身可以存放在任何主存区间，但它的入口地址被写入一个固定的单元，如写入 1 号单元，这一点在硬件上是固定的。各个中断服务程序的入口地址则被写进查询程序之中。

有些参考文献将非向量中断方式称为单向量中断，含义是这种响应方式由 CPU 指向一个固定的向量地址（在 130 机中，这个向量地址即 1）。本书主张采用非向量中断这一提法，以免混淆。

查询方式可以是软件轮询（即分设备地逐个查询有关状态标志），也可以先通过硬件取回被批准中断源的设备码，再通过软件判别，而优先排队电路则提供优先设备的设备码。

可见，非向量中断方式是通过软件方式确定中断源，再分支进入相应的服务程序处理的。这种方式可以简化硬件逻辑，灵活地修改优先顺序，但它的响应速度较慢。

现代计算机大多具备向量中断功能，但非向量中断方式可以作为一种补充手段。

2. 响应中断的条件

针对可屏蔽的中断请求，满足下述几个条件，CPU 才能响应中断。

- ① 有中断请求信号发生，如 IRQ_i 或 $INT\ n$ 。
- ② 该中断请求未被屏蔽。
- ③ CPU 处于开中断状态，即中断允许触发器 $T_{IEN}=1$ （或中断允许标志位 $IF=1$ ）。
- ④ 没有更重要的事件要处理（如因故障引起的内部中断，或是其优先权高于程序中中断的 DMA 请求等）。
- ⑤ CPU 刚刚执行的指令不是停机指令。
- ⑥ 在一条指令结束时响应（因为程序中断的过程是程序切换过程，显然不能在一条指令执行的中间就切换）。

3. 中断响应过程

不同计算机的中断响应过程可能不同。以模型机为例，在现行指令将结束时响应中断请求。例如，在现行指令的最后一个时钟周期，向请求源发出中断响应信号 $INTA$ ；并形成 $I \rightarrow IT$ 条件，在时钟周期结束时发出 $CPIT$ ，使 CPU 在执行完该指令后就转入中断周期 IT 。如前所述，中断周期是响应过程的一个专用的过渡周期，有的机器称之为中断响应总线周期。在这一周期中依靠硬件实现程序切换，需完成下面 4 项操作。

- ① 关中断。为了保证本次中断响应过程不受外界干扰，CPU 在进入中断周期后，便立即关中断（使 $T_{IEN}=0$ ）。

② 保存断点。将程序计数器 PC 的内容保存起来，一般是压入堆栈。此时，PC 内容为恢复原程序后的后继指令地址，称为断点。在低档微型计算机中，为了简化硬件逻辑，在中断周期中只将断点压栈保存，以后再在中断服务程序中保存程序状态字与有关寄存器内容。在某些高档计算机中，为加快中断处理速度，在中断周期中依靠硬件，将程序状态字也压栈保存，甚至将其他寄存器内容也一同压入堆栈。

③ 获取服务程序的入口。被批准的中断源接口通过总线向 CPU 送入向量地址（或相关编码，如中断类型码）。CPU 据此在中断周期中访问中断向量表，从中读取服务程序的入口地址（或包括服务程序状态字）。

④ 转向程序运行状态，以开始执行中断服务程序。如组合逻辑控制方式，在中断周期将要结束时形成 $I \rightarrow FT$ ，再切换到取指周期。

以上响应阶段的操作是在中断周期中直接依靠硬件实现的，并非执行程序指令，自然也不需编制程序实现，所以也常称为中断隐指令操作。CPU 设计制成后，就应具备这些功能。但编程者应了解硬、软件之间的界面，即在中断周期中 CPU 已完成了哪些操作，才知道如何在此基础上编制后面的处理程序。

5.4.5 中断处理

进入中断服务程序之后，CPU 通过执行程序，按照中断请求的需要进行相应的处理。显然，不同中断请求所要求的服务处理是不一样的。我们主要讨论一些共性问题，如保护现场、开/关中断、多重中断与单级中断、恢复现场与返回等。

为了形成完整的处理过程的概念，我们将 CPU 响应中断后所作的操作（中断隐指令操作）和 CPU 进行中断处理所作的软件操作（中断服务程序中的操作）按序列在表 5-2 中，并按多重中断方式和单级中断方式的不同进行了对比。

表 5-2 中断隐指令与中断服务程序中的操作

| | 多重中断方式 | 单级中断方式 |
|--------|-------------------------------------|-------------------------------------|
| 中断隐指令 | 关中断 保存断点及 PSW 取服务程序入口地址及新 PSW | 关中断 保存断点及 PSW 取服务程序入口地址及新 PSW |
| 中断服务程序 | 保护现场 送新屏蔽字 开中断 | 保护现场 |
| | 服务处理（允许响应更高级别请求） | 服务处理 |
| | 关中断 恢复现场及原屏蔽字 开中断 | 恢复现场 开中断 |
| | 返回 | 返回 |

1. 现场保护

执行中断服务程序时，可能使用某些寄存器，这就会破坏它们原先保存的内容。因此需要事先将它们的内容保存起来，称为现场保护。由于各个中断服务程序使用的寄存器不同，对现场的影响各不相同，因此较多的是安排在服务程序中进行现场保护。服务程序需要使用哪些寄存器，就保存哪些寄存器的原内容，一般是压入堆栈保存。

在服务程序中进行现场保护，可以根据实际需要有针对性地进行，不做无用操作。但通过执行程序来实现，速度可能较慢。因此有的计算机在指令系统中专门设置一种指令，可成组地保存

寄存器组内容，或者在中断周期中直接依靠硬件快速实现现场保护。

2. 多重中断与单级中断

在编制中断服务程序时，可以根据需要选择如下两种策略之一。

(1) 允许多重中断的处理方式

这种方式允许在服务处理过程中响应、处理优先级别更高的中断请求。这就可能形成一种中断嵌套关系，如图 5-21 所示。

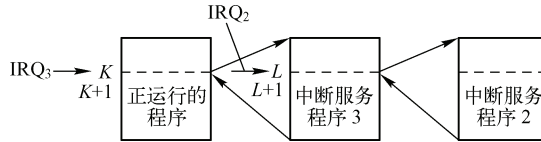


图 5-21 多重中断嵌套示意图

假定 CPU 在执行第 K 条指令时接到中断请求 IRQ_3 ，其优先级高于现执行程序。则 CPU 在执行完第 K 条指令后，转入中断周期，将断点 $K+1$ 压栈保存，然后转入中断服务程序 3。在执行服务程序 3 时，又接到优先级更高的中断请求 IRQ_2 ，于是 CPU 暂停执行服务程序 3，转入中断周期，将断点 $L+1$ 压栈保存，然后转入中断服务程序 2。当执行完服务程序 2 时，从栈中取出返回地址 $L+1$ ，返回服务程序 3。在执行完服务程序 3 之后，从栈中取出返回地址 $K+1$ ，返回到原程序，这种方式称为多重中断，大多数计算机都允许多重中断嵌套，使紧迫的事件能够得到及时处理。究竟是哪一个是先出现，这是随机的，图 5-21 中是 IRQ_3 先出现 IRQ_2 后发生， IRQ_2 比 IRQ_3 优先级高。

为了允许多重中断，在编制中断服务程序时，采取下述的安排方法。在保护现场后，送出新的屏蔽字，该屏蔽字将屏蔽掉与本请求同一优先级别以及更低级别的其他请求，然后开中断，再开始本请求源所要求的服务处理（注意，在中断周期中已由硬件关中断，为了能响应新的更高级别请求，服务程序需执行开中断指令）。如果在处理过程中，CPU 又接到优先级更高的新请求，就可以暂停正在执行的服务程序，保存其断点，转去响应新的中断请求。如果在处理过程中并无新的中断请求，则服务可以完整地执行，不被打断。

(2) 只允许单级中断的处理方式

如果响应某个中断服务请求后，CPU 只能为该请求源服务，不允许被其他中断请求所打断，只有当本次中断服务全部完成并返回到原程序后，CPU 才能重新响应新的中断请求，这种情况称为单级中断。

如果只允许单级中断，则在编制中断服务程序时采取如下的安排方法。在保护现场后即开始实质性的服务处理。由于在中断周期中已由硬件关中断，所以在本次服务过程中不再响应新的中断请求。直到本次服务完毕，临返回之前才开中断。

3. 恢复现场、返回到原程序

当服务程序完成处理任务即将返回到原程序时，应使 CPU 的有关状态恢复到被中断之前，为此应当恢复现场与打开允许中断触发器。

在恢复现场时不允许被打扰，CPU 应处于关中断的状态。对于多重中断方式，此时应暂时关中断，再恢复现场。对于单级中断方式，处理过程本来就处于关中断状态。可见在编制中断服务程序时应遵循一个原则：在响应过程、保护现场、恢复现场等过渡状态中，应当关中断，使之不受打扰。

当现场恢复之后，执行开中断指令，然后执行返回主程序指令。开中断指令一般在完成开中断操作后，立即转入下一条指令。也就是说，在返回到原程序后（断点值已送入程序计数器 PC）才会响应新的请求。

4. 中断处理程序举例

现以打印机中断服务程序为例，说明中断处理程序的基本内容。在调用打印机的主程序中，先将打印内容送入打印输出缓冲区。在开中断的前提下（一般在上电初始化后即已开中断），用输出指令启动打印机；如果在打印机工作过程中主机有事可干的话，继续安排运算处理等程序。

单独编写打印机中断服务程序模块，其流程结构如图 5-22 所示，将入口地址及程序状态字写入到中断向量表中的对应单元中。

本服务程序按允许多重中断方式编写。首先，保护现场（将本程序需要使用的寄存器的内容依次压入堆栈），然后送出新屏蔽字（禁止与打印中断同一优先级以及更低级别的中断请求），开中断（在后续程序中可以响应更高级别的请求）。

在服务处理中，先调回打印机接口的状态字，判定该打印机可否使用；判断输出缓冲区地址指针，若指针指向缓冲区首址，说明打印输出尚未开始；本次中断请求是由于打印机准备好（初始化完成）后的第一次请求，于是向接口送出一个打印字符代码，并修改地址指针。若 CPU 从接口收到一个回答信号 ACK（确认），表明接口已经接收一个字符代码，CPU 可以继续送下一个字符代码，直至送出一行打印信息为止。于是，服务程序关中断，恢复现场及原屏蔽字，再开中断，返回到原程序继续执行。此时，打印机进入打印状态，CPU 与打印机可以并行工作。

当打印机打印完一行字符后，再次提出中断请求，CPU 又进入打印中断服务程序。在完成有关例行操作后，对缓冲区地址指针做出判断。若指针指向的单元仍在缓冲区中，说明打印尚未完成，继续输出打印信息，并修改指针。若指针指向缓冲区底部，说明本次调用打印的任务已经完成，于是让打印机接口复位，结束调用。之后恢复现场、开中断、返回到原程序。

本服务程序采用这样一种方式：打印机每提出一次中断请求，CPU 响应后输出一行打印信息；打印完一行后，再提出中断请求。在打印一行时，CPU 可以继续执行程序。这种安排减少了响应请求的次数，因此可减少系统开销。有些计算机，如单用户工作方式的个人计算机，除了等待打印外并无其他事可干，每打印完一个字符代码就提出一次中断请求；每响应处理一次请求，送出一个打印字符代码。可见，每次处理所完成的操作长短应该具体问题具体分析，视实际需要而定。

图 5-22 所举之例仅说明一个服务处理程序的基本内容，实际的打印中断服务程序很可能还要

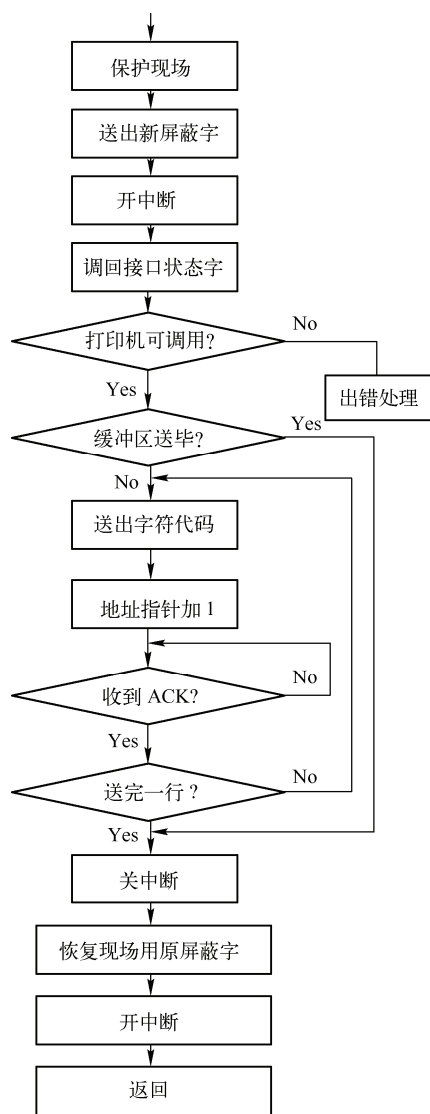


图 5-22 打印中断处理程序示例

考虑一些细节问题，这里不再详述。至此，我们将中断过程分成请求、优先排队、响应、处理、返回等几个阶段，并沿此线索讨论了有关的概念与可能的技术。

5.4.6 中断接口组成模型

中断接口是支持程序中中断方式的 I/O 接口，位于主机与某台外部设备之间。它的一侧面向系统总线，另一侧面向某台外部设备。不同的主机、不同的设备、不同的设计目标，其接口逻辑可能不同，这决定了实际应用的接口的多样化。

我们先构造一个中断接口的组成模型，来体现中断接口的基本组成原理；再以此为基础，讨论实际接口的可能变化；然后举出一个实际的中断接口，予以印证。希望这种叙述方式有助于读者掌握基本原理，举一反三，不致局限于个别特定的接口。

图 5-23 是一种在寄存器级上抽象化了的接口基本组成的功能模型粗框图，其中虚线以上是一个设备的接口，虚线以下是各设备公用的公共接口逻辑部分。

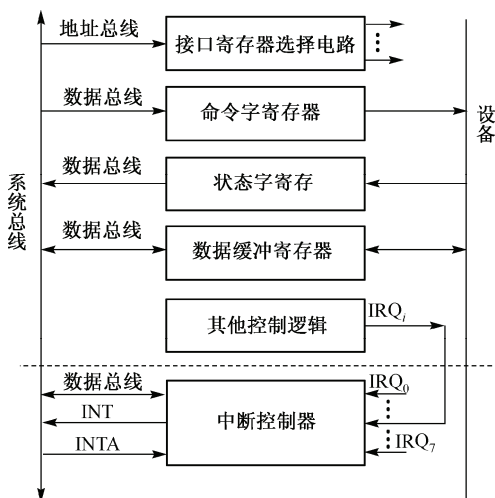


图 5-23 中断接口组成模型

在一台计算机的 I/O 子系统中，可能有多个外围接口，每个接口中又可能有数个与系统总线相连接的寄存器（或寄存器级部件，如输入通道、输出通道等），因此每个接口中需要一个选择电路，实际上是一个译码器。选择电路接收从系统总线送来的地址码，译码后产生选择信号，用于选择本接口中的某个寄存器。选择电路的具体组成与 I/O 系统的编址方式有关，因而有以下两种主要方式可供选择。

（1）统一编址方式

计算机系统将接口中的有关寄存器与主存储器统一编址，像访问主存一样地访问接口中的寄存器。相应地，为接口中的有关寄存器分配地址总线代码，寄存器选择电路对地址总线代码译码，形成选择信号，选择某个寄存器。在统一编址方式中，CPU 使用通用数据传输指令访问接口，实现输入/输出。根据地址码的范围，可区分访问主存还是访问外围接口。某些单片机系列采用统一编址方式。

（2）单独编址方式

在 IBM PC 中，用地址总线的低 8 位送出 I/O 端口地址，共 256 种代码组合，每个接口视其需要可占用一至数个端口地址。端口地址直接定位到接口中的某个寄存器，一个接口占用端口地址数可多可少，因而这种编址方式更为灵活方便。接口中的寄存器选择电路根据 I/O 端口地址译码，产生选择信号。与访问主存的总线地址不同，I/O 端口地址是专为访问外围接口设置的。

在单独编址方式中，CPU 只能通过显式的 I/O 指令访问外围接口。许多接口常将端口地址与 IOR、IOW 命令一道译码，直接形成对某个寄存器的读/写命令，既是寄存器选择，又包含了读/写控制。

2. 命令字寄存器

每种外部设备往往有自己的特殊操作，如让磁带机正转越过 n 个数据块，或者反转越过 n 个

数据块。在各种应用场合中更是常有这类情况，如让某加热炉升温或降温，等等。而通用计算机的指令系统是通用的，并不针对特殊操作，因此接口需要将通用指令转换成设备所需的特殊命令。为此，可在接口中设置一个命令字寄存器，按信息数字化的思想，事先约定命令字代码中各位的含义，或是分段译码的含义。例如，约定命令字最低位 D_0 为启动位，为 1 启动磁带机，为 0 关闭磁带机；约定 D_1 为方向位，为 1 正转，为 0 反转；约定 $D_2 \sim D_4$ 为越过 n 个数据块，如此等等。CPU 给出命令字寄存器所对应的端口地址，用输出指令从数据总线送出某个约定的控制命令字到接口的命令字寄存器，接口再将命令字代码转换为一组操作命令，送往设备。为便于调回分析，可采用双向连接。

3. 状态字寄存器

采用程序控制的一个特点是可以根据实际运行状态做出动态调整。为此，接口中常设置一个状态字寄存器，用来记录、反映设备和接口的运行状态，作为 CPU 执行 I/O 程序时的重要判别依据之一。

外围设备与接口的工作状态可以采取抽象化的约定与表示方法，如以前提到过的忙 (B)、完成 (D)、请求 (IRQ) 等。也可采取具体的描述，如设备故障、校验出错、数据迟到等一类的状态信息。

在设备与接口的工作过程中，将有关状态信息及时地送入状态字寄存器，或采取 R、S 端置入方式，或采取由 D 端同步输入方式，视具体的状态信息产生方式而定。

4. 数据缓冲寄存器

I/O 子系统的基本任务是实现数据的传输，由外部设备经接口输入到主机，或由主机经接口输出到外部设备。为此，在接口中应设置一定容量的数据缓冲寄存器。如果该寄存器只担负输入缓冲或只担负输出缓冲，则可采用单向连接模式；如果既可输入又可输出，则采取双向连接模式。

由于主机与外部设备的数据传输速度往往不同，一般地讲，主机速度应快于外部设备的速度。数据缓冲寄存器的任务就是实现缓冲，达到速度匹配。其缓冲容量称为缓冲深度，这是接口设计中必须满足的一个要求。为此，接口可能设置数个数据缓冲寄存器，甚至采用半导体存储器 SRAM 芯片构成缓冲存储器。

5. 其他控制逻辑

上面的三种寄存器是接口中基本信息的存储逻辑，这些信息或者传输内容，或者控制与有关操作的基本依据。为了按照程序中断方式实现 I/O 控制，以及针对设备特性的操作控制，接口中还需有相应的控制逻辑。这些控制逻辑的具体组成，视不同接口的需要而定，也不太规整，在组成模型框图中没有具体描述。下面列举一些可能的内容。

① 中断请求信号 IRQ 的产生逻辑。

② 与主机之间的应答逻辑。

③ 控制时序。例如，在串行接口中需有一套移位逻辑，实现串行数据传输的串并转换，相应地需有自己的控制时序，包括振荡电路、分频电路等。

④ 面向设备的某些特殊逻辑。例如，许多外围设备具有机电性操作，如电动机的启动、停止、正转、反转、加速、减速，电磁铁、继电器的动作，磁记录的编码与译码等。有些控制功能由设备控制器实现，有些则由接口负责。又如，设备的信号电平与主机不同，由接口进行电平转换等。在各种应用系统中，更需考虑这类面向设备的特殊要求。

⑤ 智能控制器。在功能要求比较复杂的接口中，常使用通用的微处理器、单片机或专用的微控制器等芯片，与半导体存储器构成一个可编程控制器。由于可以编程处理复杂的控制，常称为智能控制器型接口。

6. 公用的中断控制器

如前所述，在微型计算机系统中广泛使用集成化的中断控制器芯片（如 Intel 8259A）的任务是：汇集各接口的中断请求信号，经过屏蔽控制优先排队，形成送往 CPU 的中断请求信号 INT；在接到 CPU 批准信号 INTA 后，通过数据总线送出向量地址（或中断类型码）。这是各中断接口的公用部分，可称为公共接口逻辑，一般组装在主机板上。图 5-23 中将它画在虚线之下，一是与各设备接口从组装角度区分开来，二是可与设备接口连成一个整体，形成完整的接口逻辑概念。

针对基本接口的组成模型，以抽象化的方式对接口的工作过程描述如下。

① 初始化中断接口和中断控制器。CPU 通过调用程序或系统初始化程序，对中断接口初始化（如设置工作方式，初始化状态字、屏蔽字），为各中断请求源分配中断类型码（或其他相应的向量编码）等。

② 启动外部设备。通过专门的启动信号或命令字，使接口状态为 $B=1$ （忙标志位）、 $D=0$ （完成标志位）。据此启动设备工作。

③ 设备提出中断请求。当设备准备好，或完成一次操作，使接口状态变为 $B=0$ 、 $D=1$ 。据此向中断控制器发出中断请求 IRQ_i 。

④ 中断控制器提出中断请求。 IRQ_i 送中断控制器 8259A，经屏蔽控制和优先排队，向 CPU 发出公共请求 INT，并形成中断类型码。

⑤ CPU 响应。CPU 向 8259A 发回批准信号 INTA，并通过数据总线从 8259A 取走对应的中断类型码。

⑥ CPU 在中断周期中执行中断隐指令操作，进入中断服务处理程序。有关 8259A 的使用细节，将在后续课程中介绍。

与如图 5-23 所示的接口模型相比，实际应用中的接口可能存在以下两种变化。

（1）命令/状态字的简化

有的接口所需要的命令/状态信息不多，可以合为一个寄存器，称为命令/状态字寄存器。其中有些位可由 CPU 编程设置，体现主机向设备与接口发出的控制信息。有些位用于记录设备与接口的运行状态，据此反映状态信息。有些接口没有明显的命令/状态字，因而可进一步简化为几个触发器。例如，在 DJS-130 机基本中断接口中只设置 4 个触发器，体现基本的命令/状态信息：工作触发器 C_{GZ} （相当于忙触发器 B），结束触发器 C_{JS} （相当于完成触发器 D），中断请求触发 C_{QZ} （即 IRQ ），屏蔽触发器 C_{PB} （即 IM）。当 CPU 发清除命令时，清除信号使 $C_{GZ}=0$ 和 $C_{JS}=0$ 。当 CPU 发启动命令时，启动信号使 $C_{GZ}=1$ 和 $C_{JS}=0$ 。当设备准备好或完成一次操作时，使 $C_{GZ}=0$ 、 $C_{JS}=1$ 。根据 $C_{JS}=1$ 、 $C_{PB}=0$ 的条件，使请求触发器 $C_{QZ}=1$ ，从而向 CPU 发出中断请求信号。

在 DJS-130 机的实际外围接口中，也往往根据需要，在上述的基本接口中增加一些逻辑电路（与外围设备的具体特性有关）。

（2）命令/状态字的具体化与扩展

许多外围接口是为了连接常规外部设备而设置的，如键盘接口、打印机接口、显示器接口和磁盘接口等。这就需要针对设备的具体要求，将命令字和状态字具体化。例如，对磁带机发出的命令中，可能包含正转、反转、越过 n 个数据块、读、写等。可以按照信息数字化的基本思想，

分别确定命令字和状态字的位数，以及每一位代码的约定含义。事实上，我们从一些典型系统的成熟接口设计中可以找到借鉴的参考蓝本，本章将介绍打印机接口和磁盘接口实例。

在构成计算机应用系统时，有些接口连接的是广义外围设备，其变化可能更多。

【例 5-1】 用计算机控制某 n 层楼房的电梯系统，在主机与电梯的电动机驱动系统中应设置一个接口。主机对电梯所发出的命令中可能包含启动、停止、上升、下降、加速、减速等；作为控制依据的电梯状态中可能包含：到达 $1\sim n$ 层的请求、 $1\sim n$ 层提出的使用电梯请求等；通过接口传输的数据可能包含电梯所处位置、电梯升降速度等。因此，电梯系统接口设计的任务之一是将上述信息用约定的数字代码表示，并根据要求确定命令字、状态字、数据等寄存器的位数。

【例 5-2】 用计算机控制 n 台电热炉，启动或关闭它们，定时采集它们的炉温数据，并及时地进行反馈调节。一种可能的方案是通过定时时钟中断，激活数据采集及控制调节程序。在系统中，时钟中断作为一个中断源；但在它所引发的中断处理程序中，需分别控制处理 n 台电热炉，这可视作为一种设备数量的扩充。为此，在接口中可设置一个命令字寄存器，分为 n 段，每段的若干位代码按约定格式表明主机向对应炉号的电热炉发送的控制命令，而状态字寄存器的设计也与此相类似。

上述例子中涉及命令/状态字信息的复杂化与具体化及设备数量扩充等问题，处理的基本思路是充分利用信息表示的数字化（即用约定的数字代码格式去表达主机发出的控制命令，及外部设备与接口本身的各种状态），通过数据总线送出或回收，从而实现各种情况下的 I/O 控制。

5.4.7 典型中断接口举例

下面举几个中断接口实例，作为如图 5-23 所示的中断控制模型的实际体现。

1. 打印机中断接口举例

在打印机子系统中，常将打印机控制器制作在打印机之内，即将设备控制器与设备做成一个整体。因此，打印机与主机之间的接口（适配器）可采用通用的并行中断接口，也可用作其他设备的并行接口。

图 5-24 是一种在微型计算机系统中使用的并行接口，常用作针式打印机与主机之间的接口（称为打印机适配卡），也可用于通用的并行接口。它的一侧与系统总线相连，因此它的信号线与印制插头需要符合 PC 总线标准；接口的另一侧，通过一个 25 芯插口与针式打印机连接。在微机中，针式打印机是一种标准的外设配置，因此对 25 芯连接插件也进行了相应约定。

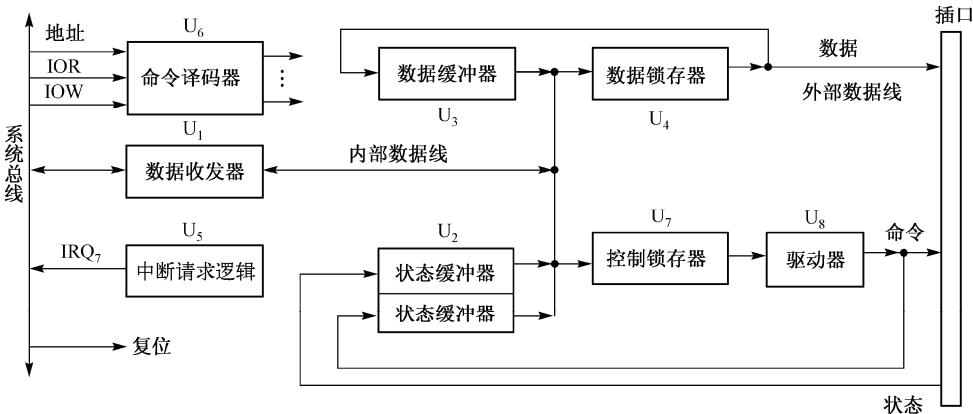


图 5-24 打印机并行接口结构

(1) 基本组成

① 命令译码器 U_6 ，相当于图 5-23 中的寄存器选择电路。

② 控制锁存器 U_7 ，相当于图 5-23 中的命令字寄存器。它的输出通过驱动器 U_8 ，经插口再送往打印机。

③ 状态缓冲器 U_2 ，其中 U_2 (上)相当于图 5-23 中的状态字寄存器，由插口获得打印机状态信息； U_2 (下)是控制命令字的回送信息缓冲，供 CPU 调回，判断所发出的命令信息在接口中是否正确。

④ 数据锁存器 U_4 ，相当于图 5-23 中的数据缓冲寄存器。打印机是输出设备，所以 U_4 的任务是将打印数据经插口送往打印机。 U_4 中的内容也可经数据缓冲器 U_3 回送 CPU，判断所输出的打印信息在接口中是否正确。

⑤ 数据收发器 U_1 ，是一个可双向收发数据的部件。一侧与系统总线相连，另一侧通过接口内部数据线，将系统总线送来的命令信息、打印信息等送入有关寄存器；或接收状态信息、回送打印信息、回送命令信息等，经系统总线再送入 CPU 供判断使用。

⑥ 中断请求逻辑，相当于图 5-23 中其他控制逻辑的部分功能，它产生中断请求信号。在微机中，约定打印机中断请求为 IRQ_7 ，优先级较低。

(2) 命令译码

命令译码器 U_6 接收系统总线送来的端口地址与读写命令 (IOW、IOR)，译码输出下述命令，选择接口中的有关寄存器：① 写输出数据寄存器 WPA (U_4)；② 写输出控制寄存器 WPC (U_7)；③ 读输入数据缓冲器 RPA (U_3)；④ 读状态寄存器 RPB (U_2 (上))；⑤ 读输入信号缓冲器 RPC (U_2 (下))。

(3) 控制命令

控制命令字的传输路线是：CPU → 系统总线 → 收发器 U_1 → 控制锁存器 → 驱动器 → 打印机及状态缓冲器 U_2 (下)。

当本接口用作打印机的接口时，约定命令字占用数据总线低 5 位传输，即 $D_4 \sim D_0$ 。其命令字的含义为：① 允许中断请求 INTEN，允许适配器向 CPU 提出中断请求 IRQ_7 ；② 选中输入信息 SLCTIN，将数据输入打印机；③ 初始化 \overline{INIT} ，对打印机进行初始化；④ 自动走纸 AUTOFDXT，使打印机自动走纸一个行距；⑤ 数据选通 STROBE，作为打印机接收数据选通工作脉冲。

(4) 状态信息

打印机状态信息的传输路径是：打印机 → 状态缓冲器 U_2 (上) → 收发器 → 系统总线 → CPU。

当本接口用作打印机接口时，约定状态字占用数据总线中的 $D_7 \sim D_3$ ，共 5 位，其各位的状态含义为：① 忙 \overline{BUSY} ，打印机处于忙状态时，不能接收数据；② 确认 \overline{ACK} ，打印机接收到数据字节后，回送一个 \overline{ACK} 信号，表示收到上一字节，可以接收新的数据字节；③ 纸完 PE，表示纸盘已经没有打印纸了；④ 选择 SLCT，这个信号表明所选择的打印机状态：为高电平，打印机处于联机状态；为低电平，打印机处于脱机状态；⑤ 出错 \overline{EROR} ，打印机发生某种故障。

(5) 打印数据

一方面，打印数据沿下述传输路线输出：主机 → 数据总线 → 收发器 U_1 → 数据锁存器 U_4 → 打印机 (供打印)。另一方面，由主机送出的打印数据也可沿数据锁存器 U_4 → 数据缓冲器 U_3 → 收发器 U_1 → 数据总线 → CPU 路线调回 CPU，供诊断判别之用。

打印数据是由 CPU 中的寄存器送出？还是由主存单元直接送出？这与主机的指令系统有关。如果采用本书模型机的设置方式，则可用数据传输指令直接由主存送往接口 U_4 ，所以前面泛指由主机输出打印数据。

2. 中断接口芯片举例

前面是从寄存器级讨论接口的组成原理,在具体逻辑实现中则应尽量采用集成化的通用接口芯片。这类接口芯片有许多产品可供选用,后续课程将会作进一步的介绍。本节仅举出三个常用芯片实例,简要说明它们的组成原理,具体的使用方法可查手册。

(1) Intel 8212 (8 位 I/O 接口芯片)

图 5-25 是一种简单的 8 位 I/O 接口芯片 Intel 8212,曾广泛用于 8 位微型计算机系统中。它的内部组成很简单,一个 8 位的数据锁存器,一个中断请求触发器。所以它提供的功能仅有简单数据传输(单向)、中断请求。从外部特性看,8212 的引脚功能如下:

- ◎ DS_1 、 DS_2 —设备选择信号(有几种不同的接法供选择)。
- ◎ MD —模式控制(即工作方式选择,控制输出缓冲器的状态,以及时钟来源)。
- ◎ C —时钟输入。
- ◎ STB —输入选通。
- ◎ CLR —清除。
- ◎ D_I —数据输入,8 位。
- ◎ D_O —数据输出,8 位。
- ◎ \overline{INT} —中断请求信号线。

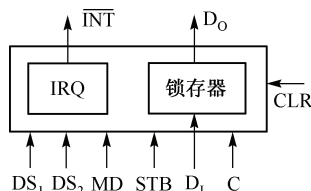


图 5-25 Intel 8212 接口芯片

(2) Intel 8255 (通用并行接口芯片)

图 5-26 是一种通用的 8 位并行接口芯片 Intel 8255,该芯片曾广泛用于微机系统中,主要由以下几部分组成。

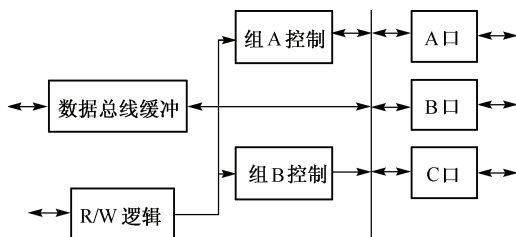


图 5-26 Intel 8255 接口芯片

① R/W 控制逻辑。接收 CPU 发来的端口地址与读写命令 RD、WR,以及复位命令 RESET;向 A 组控制、B 组控制及 A、B、C 三个端口发出操作命令。这相当于接口模型中的寄存器选择电路,以及读/写控制和复位等。

② 三个端口部件。A 口和 B 口是数据输入/输出端口,每个端口部件有一个 8 位数据输入锁存器和一个 8 位数据输出锁存器。A 口和 B 口分别使用,当成两个彼此独立的 8 位端口。C 口是控制信息端口,也是 8 位,可输入控制信号、输出端口状态信号(即提供或接收应答联络信号),如选通、中断请求、中断允许、响应输入、输入缓冲器满、输出缓冲器满等。C 口在模式控制下可以被分为 2 个 4 位口,分别与 A 口和 B 口联合工作,以控制信号的输出和状态信号的输入。

③ 组 A 控制、组 B 控制,分别控制 A 口和 B 口的控制部件,可由 CPU 编程设置,按位控制 A 口和 B 口的各位是输入或是输出。此外,组 A 和组 B 控制还可以分别控制 C 口的高 4 位组和低 4 位组,以实现更复杂的功能。

④ 数据总线缓冲,相当于打印接口(见图 5-24)中的收发器,一侧与数据总线连接,另一侧与 8255 的内部总线(芯片内)连接。从数据总线获得数据,分送片内各部件;或接收片内部件数据,送往数据总线。

因此,8255 提供了 I/O 接口的基本功能,可用来构成简单接口。如果要在接口中设置控制/状态字等,可增设一些寄存器部件。

(3) Intel 8250 (可编程异步通信串行接口芯片)

图 5-27 是一种串行接口芯片,广泛应用于微机系统的异步通信。芯片包含接收部分和发送部

分，接收部分实现串-并转换，而发送部分实现并-串转换（两部分组成相似，图中只画出接收部分的寄存器级组成）。

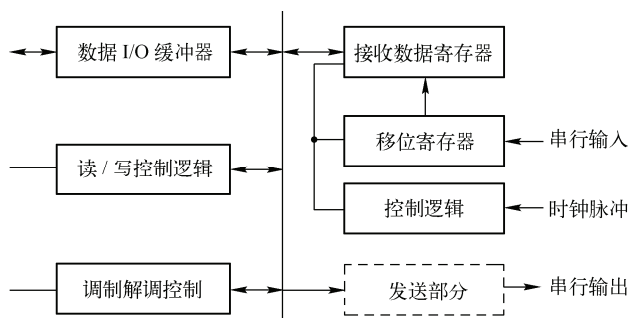


图 5-27 Intel 8250 接口芯片

① 读/写控制逻辑、数据 I/O 缓冲器，与 Intel 8255 的对应部分相似。

② 接收部件。这部分的基本功能是接收串行输入数据，经串-并转换为 8 位数据，由数据 I/O 缓冲器并行送往数据总线。串行输入采取某种串行通信标准，如 RS-232C 标准（有关概念在后续课程中介绍）。

串-并转换由移位寄存器实现：逐位串行输入，移位组装成 8 位字节，然后并行送往一个接收数据寄存器。为此，需要一个移位控制逻辑，按约定频率输入时钟脉冲，分时识别串行输入数据信号电平是高或低，并移位输入。

③ 发送部件。其组成与接收部件十分相似，但它能实现相反的转换过程。输出数据由数据总线并行送入一个发送数据寄存器，再送往移位寄存器，然后在约定时钟频率控制下进行移位输出，形成串行输出信号。

④ 调制解调控制。Intel 8250 芯片常用于串行通信中。为了实现长距离传输，需由调制器将串行输出的发送数据信号调制为高频载波信号，才能通过通信线路（如长途电话线）送往远方。从通信线路接收到的载波调制信号，需经解调器转换为数据信号，才能输入到 8250 芯片。通信设备中有专门的调制解调器（Modem），8250 为此提供了相应配合的控制功能。

从前述内容可见，串行接口逻辑一般比并行接口逻辑更复杂。

5.5 DMA 方式与接口

虽然中断方式要比程序查询方式更有效，但在实现数据传输时仍是以执行中断服务程序的方式实现的，传输效率仍不高，特别是大批量的数据传输时。因此对于大量数据的传输，需要更有效技术，如采用 DMA 控制方式。本节将从接口的角度进一步讨论实现 DMA 方式的 I/O 传输、在 I/O 子系统中的有关组成，并举例说明 DMA 的操作过程。

5.5.1 DMA 方式基本概念

1. 定义

DMA，即直接存储器访问（Direct Memory Access），几乎是所有计算机系统都具备的一种重要工作机制，是指这样一种传输控制方式：依靠硬件直接在主存与外围设备之间进行数据传输，在传输过程中不需要 CPU 的干预。

直接存储器访问（DMA）意味着在主存储器与 I/O 设备之间有直接的数据传输通路，不必经

过 CPU，也称为数据直传。也就是说，输入设备的数据可经系统数据总线直接输入到主存储器；而主存中的数据可经数据总线直接输出给输出设备，所以也称为直接存储器存取。定义的又一层含义是，这样的数据直传是直接由硬件控制实现的，不依靠执行程序指令来实现，所以在 DMA 传输期间不需要 CPU 执行程序来控制干预。

我们再简要回顾一下另外两种 I/O 传输的控制方式。在程序查询方式（直接程序传输方式）中，当条件具备时，CPU 执行 I/O 指令，发出有关微操作命令，实现数据的输入或输出。有些计算机的访存指令与 I/O 指令是分别设置的，输出时需要先执行访存指令，将数据从主存读入 CPU，再执行输出指令将数据由 CPU 写入 I/O 设备；或者反过来实现数据输入。在程序中断方式中，首先切换到中断服务程序，在该程序中执行 I/O 指令，实现数据的输入或输出。

2. 特点与应用

根据 DMA 方式的定义，这种传输控制方式的特点是以响应随机请求的方式，实现主存与 I/O 设备间的快速数据传输；DMA 传输周期的插入不影响 CPU 程序的执行状态，除非 CPU 和 DMA 访问主存引起冲突，否则 CPU 可以继续执行自己的程序，因而显著提高了 CPU 利用率；但是，也正因为无 CPU 参与，因而 DMA 方式只能处理简单的数据传输。

与程序查询方式相比，DMA 方式也可以像中断那样响应随机请求。当传输数据的条件具备时，接口提出 DMA 请求，获得批准后，占用系统总线进行数据的输入/输出，CPU 不必为此等待查询，可以并行地执行自身的程序。传输的实现是直接由硬件控制的，CPU 不必为此执行指令，其现执行程序也不受 DMA 影响（除非访存冲突）。

与程序中断方式相比，DMA 方式仅需占用系统总线，CPU 不必切换程序，不存在保存断点、保护现场、恢复现场、恢复断点等操作，因而在接到随机请求后，可以快速插入 DMA 传输。从原理上讲，只要不存在访存冲突，CPU 也可与 DMA 传输并行地工作。仅仅依靠硬件，可以实现简单的数据传输，但难以识别和处理复杂事态。

鉴于以上特点，DMA 传输方式一般应用于主存与高速 I/O 设备之间的简单数据传输。高速 I/O 设备包括磁盘、磁带、光盘等外存储器，以及其他带有局部存储器的外围设备、通信设备等。

对磁盘的读/写是以数据块为单位进行的，一旦找到数据块起始位置，就将连续地读/写。找到数据块起始位置是随机的，相应地，接口何时具备数据传输条件也是随机的。由于磁盘读写速度较快，在连续读写过程中不允许 CPU 花费过多的时间。因此，从磁盘中读出数据或向磁盘中写入数据时，一般采用 DMA 方式传输：直接由主存经数据总线输出到磁盘接口，然后写入盘片；或由盘片读出到磁盘接口，然后经数据总线写入主存。

当计算机系统通过总线与外部通信时，常以数据帧为单位进行批量传输。何时引发一次通信，可能是随机的。开始通信后，常以较快的数据传输速率连续传输，因此适合采用 DMA 方式。在不通信时，CPU 可以照常执行程序，在通信过程中仅需占用系统总线，系统开销很少。

大批量数据采集系统中也可以采用 DMA 方式。为了提高半导体存储器芯片的单片容量，许多计算机系统选用动态存储器 DRAM，并用异步刷新方式安排刷新周期。刷新请求的提出，对主机来说是随机的。DRAM 的刷新操作是对存储内容按行读出，可视为存储器内部的数据批量传输。因此，也可采用 DMA 方式实现，将每次刷新请求当成 DMA 请求，CPU 在刷新周期中让出系统总线，按行地址（即刷新地址）访问主存，实现各芯片的一行刷新。利用系统的 DMA 机制实现动态刷新，简化了专门的动态刷新逻辑，提高了主存的利用率。

DMA 传输是直接依靠硬件实现的，可用于快速的数据直传，传输过程不需 CPU 参与。也正

是由于这点，DMA 方式不能处理复杂事态。因此，在某些复杂场合常将 DMA 与程序中断方式相结合，二者互为补充。典型的例子是磁盘调用，磁盘读写采用 DMA 方式进行数据传输，而对寻道正确性的判别、批量传输结束后的处理，则采用中断方式。

3. 单字传输方式与成组传输方式

每提出一次 DMA 请求，将占用多少个总线周期？是单字传输还是成组连续传输？如何合理地安排 CPU 访存与 DMA 传输中的访存？这是系统设计时应该考虑的问题。采用 DMA 方式是为了实现一次批量传输，如从磁盘中读出一个文件，但在实施上可以有两种方案。

(1) 单字传输方式

每次 DMA 请求获得批准后，CPU 让出一个总线周期的总线控制权，由 DMA 控制器控制系统总线，以 DMA 方式传输一字节或一个字（如一次并行传输 8 位、16 位、32 位等）。结束后，DMA 控制器归还总线控制权，CPU 再重新判断下一个总线周期的总线控制权是 CPU 保留，还是继续响应一次新的 DMA 请求。这种方式称为单字传输方式，又称为周期挪用或周期窃取，即每次 DMA 请求都从 CPU 控制时间中挪用一個总线周期，用于 DMA 传输。

当主存储器工作速度高出 I/O 设备较多时，采用单字传输方式可以提高主存利用率，对 CPU 程序执行的影响较小。因此，高速主机系统常采用这种方式，这是因为在 DMA 传输数据尚未准备好（如尚未从磁盘中读得新的数据）时，CPU 可使用系统总线访问主存。根据主存读写周期与磁盘的数据传输率，可以计算出主存操作时间的分配情况：有多少时间需用于 DMA 传输（被挪用），有多少时间可用于 CPU 访存，这在一定程度上反映了系统的处理效率。由于访存冲突，每次 DMA 传输会对 CPU 正常执行程序带来一定的影响，但由于主存速度较高，单字传输方式不会造成一段死区，因而影响不严重（每次申请、判别、响应、恢复，毕竟要花费一些时间）。

(2) 成组连续传输方式

每次 DMA 请求获得批准后，DMA 控制器掌管总线控制权，连续占用若干个总线周期，进行成组连续的批量传输，直到批量传输结束，才将总线控制权交还给 CPU。这种方式称为成组连续传输方式，在传输期间，CPU 停止访问主存，处于保持状态，因此无法继续执行程序。

当 I/O 设备的数据传输率接近于主存工作速度时，或者 CPU 除了等待 DMA 传输结束并无其他事可干（如单用户状态下的个人计算机）时，常采用成组连续传输方式。这种方式可以减少系统总线控制权的切换次数，有利于提高 I/O 效率。由于系统必须优先满足 DMA 高速传输，如果 DMA 传输的速度已接近于主存速度，则每个总线周期结束时将总线控制权交回给 CPU 就没有多大意义。对单用户个人计算机，一旦启动调用磁盘，CPU 就等待这次调用结束才恢复执行程序，因此也可以等到批量传输结束才收回总线控制权。对于高速计算机，常采用多道程序工作方式，且主存速度超出 I/O 速率很多，如果采用成组连续传输方式，就会影响主机的利用率。

4. 硬件组织

我们一再强调，DMA 传输是直接依靠硬件实现的，这是它不同于程序查询方式与程序中断方式之处。那么，为了实现 DMA 传输，需要哪些硬件组成？由谁控制 DMA 传输？

DMA 方式实现主存与 I/O 设备间的数据直传。为此，应当指出传输方向，是输入还是输出；应当给出 I/O 设备的寻址信息，如磁盘的驱动器号、圆柱面号、磁头号、起始数据块号等；还要给出主存缓冲区的寻址信息，对于连续存储区来说，往往给出该缓冲区首地址（即起始地址）以及数据的交换量。

在早期的计算机中，DMA 传输是由 CPU 与 DMA 接口协同控制的。以 DJS-100 系列为例，

它将 DMA 方式称为数据通道方式，在主机与高速 I/O 设备之间设置数据通道接口。该接口中有如下寄存器：数据缓冲寄存器 J；控制/状态寄存器 A，其中存放主机送来的传输命令与外围设备的寻址信息；地址寄存器 B，在 DMA 初始化时送入主存缓冲区首址，每次 DMA 传输之后，内容加 1，指向主存缓冲区下一单元；交换字数计数器 C，在 DMA 初始化时送入批量传输字数的补码值，每次 DMA 传输之后，内容加 1，当计数器溢出时表明批量传输结束，提出中断请求以进行 DMA 传输的结束处理。此外，在接口中还设置了一些触发器，如数据通道请求触发器、数据通道选中触发器、同步触发器等。当具备数据通道传输条件时，接口向 CPU 提出数据通道请求。CPU 响应请求后，进入数据通道状态，相当于模型机中的 DMA 周期。在该状态周期中，CPU 暂停执行程序，发出取数据通道地址命令，将接口 B 寄存器内容送至地址总线；将接口 A 寄存器中的数据通道操作方式信息取回 CPU，据此由 CPU 发出微操作命令，实现 DMA 传输。因此，这种控制策略是在 DMA 初始化时通过程序将有关控制信息送往 DMA 接口；在进行 DMA 传输时，以接口提供的有关信息为依据，由 CPU 根据传输指令发出微命令，实现 DMA 传输。

现在的计算机系统中专门设置 DMA 控制器，由它控制 DMA 传输，而且较多地采取 DMA 控制器与 DMA 接口相分离的方式。DMA 控制器只负责申请、接管总线的控制权，发出传输命令和主存地址，控制 DMA 传输过程的起始和终止，因而可以通用，独立于具体 I/O 设备。DMA 接口则实现与设备的连接和数据缓冲，反映设备的特定要求。按照这种方式，DMA 控制器中存放着传输命令信息、主存缓冲区地址信息、交换量等，其功能是接收接口发来的 DMA 请求，向 CPU 申请掌管总线，然后向总线发出传输命令与总线地址，控制 DMA 传输。在逻辑划分上，DMA 控制器是 I/O 子系统公共接口逻辑，为各 DMA 接口所公用，是控制系统总线的设备之一。在具体组装上，DMA 控制器有集成芯片可供选用，常将它装配在主机系统板上。DMA 接口的组成与功能则相应简化，一般包含数据缓冲寄存器、I/O 设备寻址信息、DMA 请求逻辑。可以根据寻址信息访问 I/O 设备，将数据读入数据缓冲寄存器，或由数据缓冲寄存器写入设备。在需要进行 DMA 传输时，接口向 DMA 控制器提出请求；在获得批准后，接口将数据缓冲寄存器内容经数据总线写入主存缓冲区，或将主存内容写入接口（CPU 不参与传输过程的具体控制）。

5. 程序准备（DMA 初始化）

虽然 DMA 传输本身是直接依靠硬件实现的，但为了实现相关控制，CPU 需要事先向 DMA 控制器送出有关控制信息。在调用 I/O 设备时，通过程序所做的这些准备工作常称为 DMA 的初始化，即向 DMA 控制器和接口传输并设置初始信息。一般来说，DMA 初始化时通常要包含下述 4 步基本操作。

（1）向接口送出 I/O 设备的寻址信息。例如，要从磁盘中读出一个文件，则需送出该文件所在磁盘的驱动器号、圆柱面号、磁头号（记录面号）、起始扇区号（或数据块号）。

（2）向 DMA 控制器送出控制字，主要是数据的传输方向，输入主存还是从主存输出。

（3）向 DMA 控制器送出主存缓冲区首地址。数据的输入或输出，往往需在主存储器中设置相应的缓冲区，这是一段连续的存储区，为此在初始化时送出其首地址。

（4）向 DMA 控制器送出交换量，即数据的传输量。视设备的需要，传输量可以是字节数、字数、数据块数。

5.5.2 DMA 控制器与接口的连接

为了与常用芯片的用语相吻合，本书将 DMA 控制器定义为负责申请、控制总线以控制 DMA

传输的功能逻辑；将狭义的 DMA 接口定义为与具体设备相适配，进行数据传输的接口逻辑。这两部分组成了广义的 DMA 接口。

DMA 控制器与接口的具体组成，取决于对以下几方面的设计考虑，因而有多种方案。

- ① DMA 控制器与 I/O 接口是相互分离，还是合为一体？
- ② 数据传输是经由 DMA 控制器，还是接口直接经数据总线与主存相连？
- ③ 如果一个 DMA 控制器连接多台设备，是采取选择型工作方式，还是多路型工作方式？
- ④ 如果一个计算机系统中有多个 DMA 控制器，是采用公共 DMA 请求方式，还是采用独立 DMA 请求方式？

下面列举若干常见的连接模式。

1. 单通道型 DMA 控制器

如果一个 DMA 控制器只连接一台 I/O 设备（即只有一个通道），则 DMA 控制器与 I/O 接口就没有必要分开，可合为一个整体，称为单通道 DMA 控制器。其内部组成、与系统及设备的连接模式如图 5-28 所示。

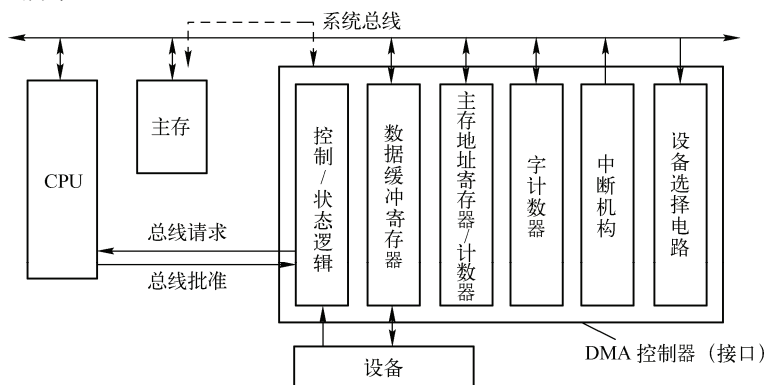


图 5-28 单通道型 DMA 控制器

- ① 设备选择电路：接收主机在 DMA 初始化阶段送来的端口地址码，经译码产生选择信号，选择 DMA 控制器内的有关寄存器。
- ② 数据缓冲寄存器：一侧与数据总线相连，另一侧与 I/O 设备相连。
- ③ 主存地址寄存器/计数器。在初始化时，CPU 将主存缓冲区首地址经数据总线送入。在 DMA 控制器掌管总线时，经地址总线送出主存缓冲区地址。每传输一次，计数器内容加 1，指向下一次传输单元。
- ④ 字计数器：在初始化时，CPU 经数据总线送入本次调用的传输量，以补码表示。每传输一次，计数器内容加 1。当计数器溢出时，结束批量传输。
- ⑤ 控制/状态逻辑。在初始化时，CPU 经数据总线送入控制字，内含传输方向信息。当具备一次 DMA 传输条件时，DMA 请求触发器为 1，控制/状态逻辑经系统总线向 CPU 提出总线请求。如果 CPU 响应，发回批准信号，DMA 控制器接管总线控制权，向系统总线送出传输命令和总线地址码。
- ⑥ 中断机构。如前所述，DMA 控制方式常与程序中断方式配合使用，所以在 DMA 接口中常含有中断机构。典型的应用是：当计数器计满溢出时，便提出中断请求，CPU 通过执行特定的中断服务程序进行结束处理。

当从设备向主存输入数据时，数据经 DMA 控制器、数据总线，直接输入到主存缓冲区，不

经过 CPU；当数据从主机向外部设备输出时，由主存缓冲区经过数据总线、DMA 控制器输出到设备，此时也不会经过 CPU。

2. 选择型 DMA 控制器

图 5-29 是选择型 DMA 控制器。在物理意义上，一个 DMA 控制器可以连接多台设备，或者说，多台设备都通过一个公用的 DMA 控制器进行 DMA 传输。在具体工作时，某一段时间内控制器只选择其中的一台设备，让它完成 DMA 传输，所以称为选择型 DMA 控制器。

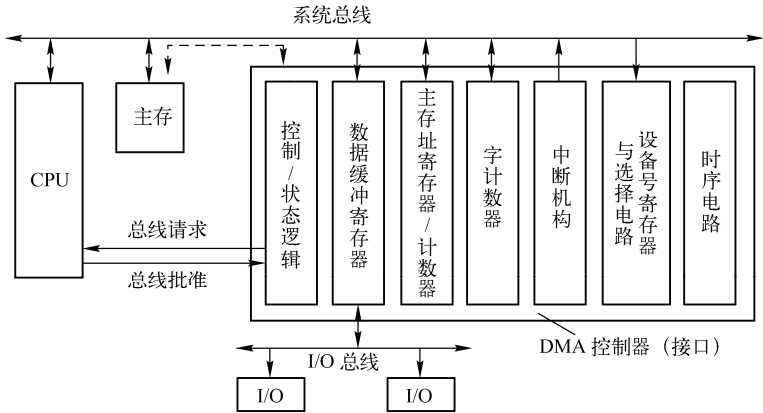


图 5-29 选择型 DMA 控制器

在逻辑划分上，这种连接模式是将大部分接口逻辑与 DMA 控制器合为一体，经由 DMA 控制器进行数据传输。各设备通过一个简单的局部 I/O 总线与 DMA 控制器相连接，某一段时间内，只有被选中的一台设备使用局部 I/O 总线。因此，设备一侧只需要简单的发送/接收控制逻辑，接口逻辑中的大部分，如数据缓冲寄存器、设备号寄存器、时序电路等都在 DMA 控制器中。此外，DMA 控制器中还包括为申请、控制系统总线所需的功能逻辑，如 DMA 请求逻辑、控制/状态逻辑、主存地址寄存器/计数器、交换字数计数器等。

在 DMA 初始化时，CPU 将所选择的设备号送入 DMA 控制器中的设备号寄存器，据此选择某台 I/O 设备。每次预置后，以数据块为单位进行 DMA 传输。当一个数据块传输完毕后，CPU 可以重新预置，选择另一台 I/O 设备。

因此，这种选择型 DMA 控制器适于数据传输率很高，以致接近于主存速度的设备，在这种情形下，不允许在批量传输中切换设备。选择型 DMA 控制器的功能相当于一个数据传输的切换开关，以数据块为单位进行选择与切换。

3. 多路型 DMA 控制器

如果所连接的多台设备速度较慢，就可以让它们同时工作，以字节或字为单位，交叉地轮流使用系统总线进行 DMA 传输，这就形成了多路型 DMA 控制器。

常见的多路型 DMA 控制器连接模式如图 5-30 所示，将 DMA 控制器与接口分离，各设备都有自己的 I/O 接口，如硬盘适配器、软盘适配器、通信适配器等。这些 I/O 接口中含有数据缓冲寄存器或小容量缓冲存储器，数据经接口与数据总线直接向主存传输，不经过 DMA 控制器（DMA 控制器负责申请并接管总线）。这样，DMA 控制器就可以通用并便于集成化，不受具体设备特性的约束。

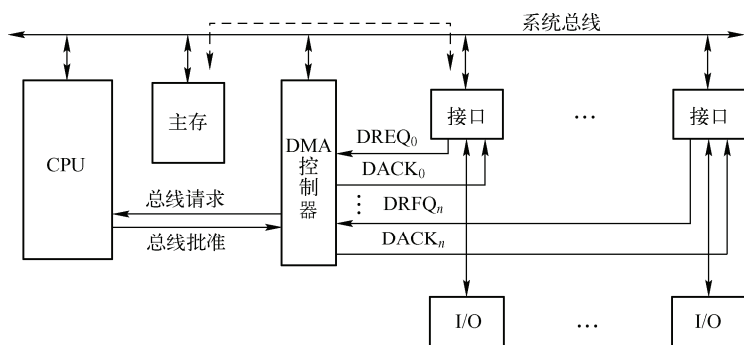


图 5-30 多路型 DMA 控制器

这样的 DMA 系统中存在两级 DMA 请求逻辑：接口中的请求逻辑与设备特性有关，在该设备需要进行 DMA 传输时，接口向 DMA 控制器提出 DMA 请求；然后，DMA 控制器向 CPU 申请占用系统总线。如果 CPU 响应，则放弃对系统总线的控制权（有关输出呈高阻抗，与系统总线脱钩），并向 DMA 控制器发出批准信号。DMA 控制器获得批准后，接管系统总线（送出总线地址与传输命令），并向接口发出响应信号。

各 I/O 接口与 DMA 控制器之间可以采取独立请求线与批准线连接方式（见图 5-30），也可采用链式连接方式，如菊花链式连接模式。

实用的多路型 DMA 控制器可以兼有选择型功能，即前述的单字传输和成组传输方式。如果采取单字传输方式，让各设备以字节或字为传输单位，交叉占用系统总线进行 DMA 传输，就是典型的多路型。如果各设备速度不同，则它们对系统总线的占有率也就不同。速度慢的设备准备一次 DMA 传输数据所需的时间长些，占用系统总线的间隔也长些；速度快的设备准备一次 DMA 传输数据所需的时间短些，占用系统总线的间隔也就短些。DMA 控制器根据各请求的优先顺序及提出的时间，随机地予以响应、分配总线周期。就系统总线而言，由各设备交叉地分时占用（但不一定均匀）。各设备则可同时工作，如将磁带上的文件复制于磁盘上，则磁带与磁盘可以同时连续读/写。

如果采取成组连续传输方式，让各设备以数据块为单位，占用系统总线进行 DMA 传输，则是典型的选择型。一个设备开始传输一个数据块，就需连续占用系统总线，中间不能打断。当该设备传输完一个数据块后，才能切换、选择另一台设备，因而这种情况属于选择型或是接近于典型的选择型。由于在一个数据块的传输过程中不允许打断，因此在系统设计时需要妥善安排优先顺序、数据块大小及 I/O 接口的缓冲深度。如果在设备 1 传输过程中，设备 2 可能提出请求，且不能耽误太久，则应将设备 1 的数据块长度安排得小些，让设备 2 接口的数据缓冲寄存器容量适当大些。

4. 多个 DMA 控制器的连接

如前所述，当一个系统需要连接多台 I/O 设备时，可以采用选择型或多路型 DMA 控制器。常用的 DMA 控制器集成芯片，其通路数量往往十分有限。如果系统规模较大，连接的设备数量较多，这时就需要采用几块 DMA 控制器芯片，然而这些芯片与系统之间应该如何连接呢？图 5-31 给出了几种常见的连接模式。

级联方式如图 5-31(a)所示，将 DMA 控制器分级相连。每个 DMA 控制器可接收多路设备请求，最后汇集为一个公共请求 HRQ。第二级 DMA 控制器的 HRQ，送往前一级 DMA 控制器的请求输入端；第一级 DMA 控制器的输出 HRQ，则送往 CPU 作为总线请求。

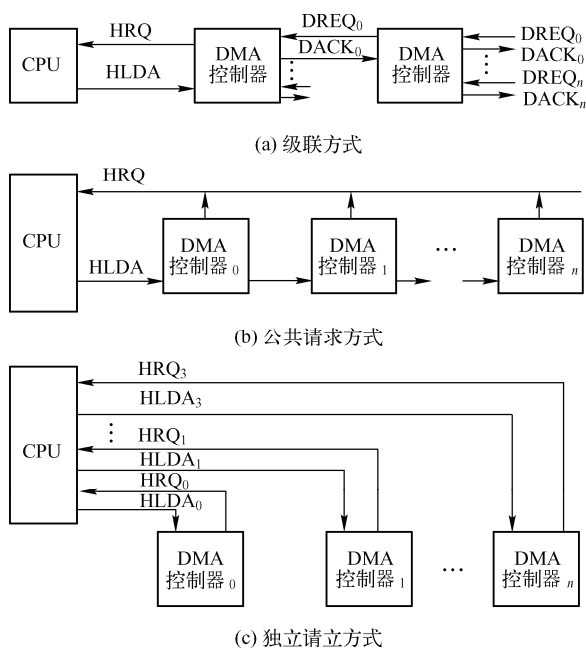


图 5-31 DMA 控制器与系统的连接方式

公共请求方式如图 5-31(b)所示, 各 DMA 控制器的请求输出 HRQ, 通过一条公用的 DMA 请求线送往 CPU, 作为总线请求。CPU 发回的批准信号, 则采用链式传递方式送给各 DMA 控制器 (按优先顺序连接)。在提出请求的 DMA 控制器中, 优先级高的先获得批准信号, 就将该信号暂时截留。待它完成 DMA 传输后, 再往后继续传输批准信号, 允许另外的 DMA 控制器获得总线控制权, 控制相应设备进行 DMA 传输。

独立请求方式如图 5-31(c)所示, 每个 DMA 控制器与 CPU 之间都有一对独立的请求线与批准线。采用这种连接模式, 取决于 CPU 是否有多对 DMA 请求输入端与批准信号输出端, 并有一个优先权判别电路 (或总线仲裁逻辑), 以确定响应当前最优先的 DMA 请求。

5.5.3 DMA 控制器的组成

在上述几种连接模式中, 较常使用的是将 DMA 控制器与 I/O 接口相分离的模式。因为 I/O 接口通常需要反映对应设备的特性, 所以常分别设置。DMA 控制器负责申请总线控制权、控制总线操作, 这些公共操作可由一个通用的 DMA 控制器实现。下面以 Intel 8237 为例, 介绍一种常用的多路型 DMA 控制器芯片的基本结构。

Intel 8237 是一种四通道的多路型 DMA 控制器芯片, 广泛应用于 IBM PC 系统中。在 PC-XT 机中只用一片 8237 (主机板上), 4 个通道按优先顺序分配给动态存储器刷新、软盘、硬盘、同步通信 (该通道可供扩展)。如果采用多片 8237 芯片级联, 还可以扩展 DMA 控制器的有效通道数目。

8237 芯片不仅支持 I/O 设备与主存之间的数据直传, 也能支持存储器与存储器之间的传输。允许编程选择的 3 种基本数据传输模式分别是: 单字节传输、数据块连续传输、数据块请求传输 (即数据块间断传输方式)。在第三种传输模式中, 当 DMA 请求 DREQ 信号有效时, 数据块连续传输; 当 DREQ 信号无效时, 暂停传输; 此信号再次有效时, 继续传输。按此方式断续传输, 直

至目标数据块传输完毕。

Intel 8237 芯片工作在 5 MHz 的时钟频率下, 数据传输率可达 1.6 MB/s, 每通道允许 64 KB 访存空间, 允许批量传输数为 64 KB, 其内部组成与外特性如图 5-32 所示。

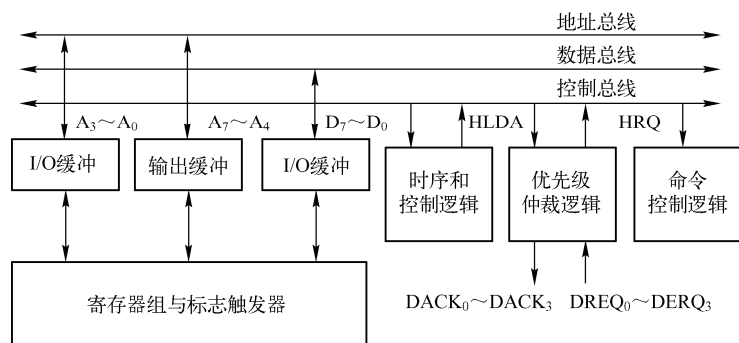


图 5-32 Intel 8237 DMA 控制器内部组成

1. 内部寄存器组

在 8237 芯片内共有 12 种寄存器和 3 种标志触发器, 用来存放 DMA 初始化时送入的预置信息, 以及在 DMA 传输过程中产生的有关信息, 作为控制总线进行控制的依据。有些寄存器是 4 个通道公用的, 有些是每个通道单独设置的。

每个通道各有一组寄存器: 基地址寄存器 (16 位), 当前地址计数器 (16 位), 基本字节数寄存器 (16 位), 当前字节数计数器 (16 位), 方式控制字寄存器 (8 位), 屏蔽标志触发器和请求标志触发器 (各 1 个)。

各通道公用的一组寄存器: 暂存地址寄存器 (16 位), 暂存字节数计数器 (16 位), 操作命令字寄存器 (8 位), 屏蔽字寄存器 (4 位), 主屏蔽字寄存器 (4 位), 状态字寄存器 (8 位), 请求字寄存器 (4 位), 暂存寄存器 (8 位), 先/后触发器 (1 个)。

① 基地址寄存器: 在初始化时由 CPU 写入主存缓冲区首地址, 作为副本保存。可在自动预置期间重新预置当前地址计数器, 这种预置不需 CPU 干预。

② 当前地址计数器: 在初始化时由 CPU 同时写入主存缓冲区首地址, 每次 DMA 传输 1 字节后, 内容加 1 或减 1 (由方式控制字选择)。因此, 它存放在在 DMA 传输期间的当前地址码 (主存缓冲区)。若选择自动预置操作, 则在 EOP 信号有效时, 该寄存器内容返回到初始值。

③ 基本字节数寄存器: 在初始化时, 由 CPU 写入需要传输的数据块字节数, 并作为初值的副本保存。如果需要重复数据块的传输过程, 可选择自动预置方式, 每当一次数据块传输结束时, 结束信号 $\overline{\text{EOP}}$ 就将副本保存的初值自动重新预置给当前字节计数器。

④ 当前字节数计数器: 在初始化时, 由 CPU 写入需要传输的数据块字节数, 一般以补码表示。每传输 1 字节, 计数器内容加 1。当一个数据块传输完毕, 计数器满, 产生结束信号 $\overline{\text{EOP}}$ 。

⑤ 方式控制字寄存器: 初始化时由 CPU 写入, 以确定该通道的操作方式。

D_1D_0 为通道选择, 因为 4 个通道的方式控制字寄存器公用一个端口地址, 故由 D_1D_0 进一步指明该方式字应写入哪一个通道的方式字寄存器。

D_3D_2 为 DMA 传输方向。

D_4 为自动预置方式选择位。

D_5 为地址增减选择位, 选择地址自动加 1 或减 1 方式。

D_7D_6 为工作方式选择: 数据块请求方式、单字节方式、数据块连续传输方式、8237 芯片级

联方式。

⑥ 暂存地址寄存器：暂存当前地址寄存器的内容。

⑦ 暂存字节数计数器：暂存当前字节数计数器的内容。

这两个寄存器与 CPU 不直接发生关系。

⑧ 操作命令字寄存器：初始化时由 CPU 写入操作命令字，指定 8237 的一些操作方式。

D₇ 选择各通道对设备的批准信号 DACK 是低电平有效，还是高电平有效。

D₆ 选择各通道设备的请求信号 DREQ 是低电平有效，还是高电平有效。

D₅ 选择是正常写入还是扩展写入。所谓扩展写入，是指写信号比正常写信号延长一倍时间，以适应慢速存储器或慢速 I/O 设备的写入操作。

D₄ 选择是固定优先级方式还是循环优先级方式。

D₃ 选择是正常时序还是压缩时序。正常时序指 DMA 周期为一个总线周期，包含 4 个时钟周期。压缩时序指一个总线周期只占 2 个时钟周期，适应快速 DMA 传输。

D₂ 允许或禁止 8237 芯片工作。

D₁ 选择通道 0 的源地址不变，或是递增、减。

D₀ 允许或禁止存储器与存储器传输。

D₁D₀ 用于控制存储器与存储器传输。在这种传输方式中，8237 规定用通道 0 保存源存储块的地址，用通道 1 保存目的存储块的地址和传输字节数。

⑨ 屏蔽字寄存器：由 CPU 送入屏蔽字，使某个通道的屏蔽标志触发器置位或复位，以确定该通道的 DMA 请求被禁止或是允许。编程时屏蔽字可为 8 位，位 D₇~D₃ 并未定义，可为 0。有效的只有低 3 位，其中 D₂ 决定对屏蔽标志触发器是置位还是复位，D₁D₀ 译码决定选择的是哪一个通道。

⑩ 主屏蔽字寄存器：采用由 CPU 送主屏蔽字的方式，可同时使 4 个通道的屏蔽标志触发器置位或复位。D₃~D₀ 这 4 位分别对应通道 3~通道 0，为 0 复位，为 1 置位。

状态字寄存器：保存状态字，供 CPU 了解各通道的工作状态。

D₇~D₄ 分别对应 4 个通道的 DMA 请求是否被处理。为 0 表示尚未处理。

D₃~D₀ 分别对应 4 个通道的 DMA 传输是否结束，为 0 表示尚未结束。

请求字寄存器：8237 允许 CPU 编程发出请求命令字，使各通道的请求标志触发器置位或复位。若某通道的请求标志被置位为 1，则该通道申请 DMA 传输，以数据块方式传输。D₂ 决定是置位或复位，D₁D₀ 选择通道。以这种方式提出的请求，称为软请求。

暂存寄存器：在存储器与存储器传输时，需要给出源地址与目的地址，但 DMA 控制器 8237 每次只能给出一个地址，因而需要占用两个总线周期。在第一个总线周期，8237 给出源地址，将数据读出并送入暂存寄存器。在第二个总线周期中，8237 再给出目的地址，将数据从暂存寄存器写入目的存储单元。

先/后触发器：由于 8237 中有些寄存器是 16 位的，而 PC/XT 机系统总线的数据通路宽度为 8 位，需分两次访问，故设置了一个先/后触发器。初值为 0，CPU 读/写低字节；然后触发器为 1，CPU 可读/写高字节；读/写完 16 位后，触发器又复位为 0。

2. 数据、地址缓冲器

这组缓冲器实现数据与地址的输入/输出，由于芯片引脚数有限，采取复用技术。

① 芯片空闲期。当 8237 尚未申请与接管系统总线控制权时，称 8237 芯片处于空闲期。在此期间 CPU 可以访问 8237，进行 DMA 初始化（预置工作方式与有关信息）；也可读出芯片内部寄存

器内容，以供判别。为此，由 CPU 向 8237 送出端口地址信息与读/写命令，并发送或接收数据。

地址输入 $A_3 \sim A_0$ ，配合读写命令 IOR、IOW，选择 8237 某一内部寄存器，读出或写入。数据输入/输出 $D_7 \sim D_0$ 经另一缓冲器实现。此时， $A_7 \sim A_4$ 未用。

② DMA 服务期。当 8237 提出总线申请、接管总线，直到 DMA 传输结束，称芯片处于 DMA 服务期。在此期间由 8237 送出总线地址，以控制 DMA 传输。此时 3 个缓冲器全部输出， $D_7 \sim D_0$ 、 $A_7 \sim A_4$ 、 $A_3 \sim A_0$ ，共输出 16 位总线地址。其中 $D_7 \sim D_0$ 送到一个地址锁存器（芯片外）。

如果是存储器与 I/O 设备间的 DMA 传输，则送出的总线地址为主存缓冲区地址。传输的另一方是设备接口中的数据缓冲器，数据直接由数据总线传输，不经过 8237。

如果是存储器与存储器间的 DMA 传输，则分两个总线周期进行，如前所述。 $D_7 \sim D_0$ 在送出总线地址高 8 位之后，又提供数据的输入/输出缓冲。

3. 时序和控制逻辑

这部分逻辑接收外部输入的时钟、片选及控制信号，产生内部的时序控制及对外的控制信号输出。

① 输入信号，包括：

CLK — 时钟输入，5 MHz。

\overline{CS} — 片选，低电平有效。

RESET — 复位，高电平有效。使芯片进入空闲期，除屏蔽寄存器被置位外，其余寄存器均被清除。

READY — 就绪，高电平有效。当选用慢速存储器或低速 I/O 设备时，需要延长总线周期，可使 READY 处于低电平，表示传输尚未完成。当完成一次传输后，让就绪信号 READY 变为高电平，通知 8237。

② 输出信号，包括：

ADSTB — 地址选通，高电平有效。将 8237 数据缓冲器送出的高 8 位地址，选通送入一个外部的地址锁存器（此后数据缓冲器就可作为数据的输入/输出缓冲用）。

AEN — 地址允许输出，高电平有效。将地址送入地址总线，其中高 8 位来自芯片外的地址锁存器，低 8 位直接来自芯片内的地址缓冲 $A_7 \sim A_0$ ，共 16 位。

\overline{MEMR} — 存储器读，低电平有效，8237 发出的控制命令。

\overline{MEMW} — 存储器写，低电平有效，8237 发出的控制命令。

③ 双向信号，包括：

\overline{IOR} — I/O 读，低电平有效。

\overline{IOW} — I/O 写，低电平有效。

在 8237 处于空闲期时，CPU 可向 8237 发出上述两个命令之一，对 8237 内部寄存器进行读/写。在 DMA 服务期中，由 8237 向总线送出上述两个命令之一，控制对 I/O 设备（接口）的读/写。

\overline{EOP} — 过程结束，低电平有效。当外部向 8237 送入过程结束信号时，将终止 DMA 传输，如由 CPU 发出结束命令。当 8237 内部的传输字节计数器计满，表明数据块传输完毕，将由 8237 向外发出 \overline{EOP} ，终止 DMA 服务。

4. 优先级仲裁逻辑

这部分逻辑实现 I/O 设备（接口）与 8237 之间的请求与响应。如果同时有两个以上设备提出请求，优先级仲裁逻辑将进行排队判优。8237 具有固定优先级、循环优先级两种优先级排队方式，可供编程选择。

如果在 DMA 初始化时指定为固定优先级方式,则优先级顺序固定,从高到低依次为通道 0~通道 3。如果选择循环优先级方式,某通道被服务一次后将降为最低优先级,其他通道依次递升。

DREQ₀~DREQ₃ — DMA 请求,由设备(接口)发来,共 4 根请求线。

HRQ — 总线请求,由 8237 发往 CPU 或其他总线控制器。

HLDA — 总线保持响应,由 CPU (或其他总线控制器)发给 8237 的响应信号。

DACK₀~DACK₃ — DMA 应答,由 8237 发往某个被批准的设备(接口),共 4 根。

5. 程序命令控制逻辑

这部分逻辑负责对 CPU 送来的命令字进行译码处理。

5.5.4 DMA 传输操作过程

下面将以 IBM PC 为例,并结合 Intel 8237 芯片的具体信号,进一步解释说明 DMA 传输的全过程。Intel 8237 的工作状态可以分为空闲周期和操作周期。操作周期又可细分为若干状态 S_i,有的状态只维持一个时钟周期,有的状态则可能维持若干时钟周期。我们以时序状态图形方式描述微机的 DMA 工作过程,如图 5-33 所示。

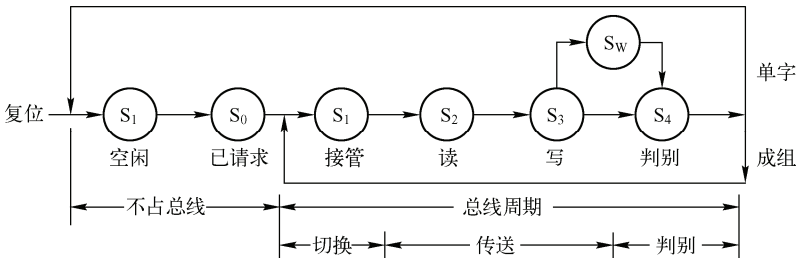


图 5-33 DMA 传输过程示例

1. S₁ 空闲周期 (静态)

CPU 可以利用 8237 处于空闲周期进行 DMA 初始化,如预置 DMA 操作方式、传输方向、主缓冲区首址、传输字节数等。同时,CPU 还应向接口送出 I/O 设备的寻址信息。在 S₁ 中,CPU 也可从 8237 读回状态字等信息,以供 CPU 判断。

8237 的每个时钟周期都采样 \overline{CS} ,看 CPU 是否选中 8237 芯片,以便对 8237 进行读/写。每个时钟周期也要采样 DREQ,看设备是否提出 DMA 请求。

当 8237 完成初始化设置以后,如果已接收到设备的 DMA 请求,则向 CPU 发出总线请求信号 HRQ,并进入 S₀ 状态。

2. 操作周期 (DMA 传输服务)

① 初始态 S₀: 此时,8237 已经发出总线请求信号,等待 CPU 的批准,如果总线正忙,8237 有可能等待若干时钟周期。当 8237 接到 CPU 发来的批准信号 HLDA 后,进入 S₁ 状态。

② 操作态 S₁: 此时,CPU 已经放弃总线控制权,8237 接管总线,送出总线地址,然后进入 S₂ 状态。所以,S₁ 是进行总线控制权切换的状态,又称为应答状态。

③ 读出 S₂: 此时,8237 向设备发出响应信号 DACK,并向总线送出读命令 \overline{MEMR} 或 \overline{IOR} 。从存储器或从 I/O 设备(接口)读出数据。

④ 写入 S₃: 此时,8237 发出写命令 \overline{IOW} 或 \overline{MEMW} ,将数据写入 I/O 设备(接口)或存储器。同时,8237 中的当前地址计数器与当前字节计数器进行内容修改(如前者加 1,后者减 1)。

⑤ 延长等待 S_w : 若在 $S_2 \sim S_3$ 内来不及完成传输 (就绪信号 $READY$ 为低电平), 则进入 S_w , 延长总线周期, 继续数据传输操作。完成一次 DMA 传输后, $READY$ 变为高电平, 8237 进入 S_4 。

⑥ 判别 S_4 : 判别 8237 采取的传输方式, 以采取相应的操作。

如果是单字节传输方式, 则 8237 结束操作, 放弃对总线的控制, 然后返回到 S_1 空闲周期。当设备再次提出 DMA 请求时, 8237 再次申请总线控制权。

如果是数据块连续传输方式, 则 8237 在完成一次传输后, 返回到 S_1 状态, 继续占用下一个总线周期, 经 $S_1 \sim S_4$ 继续传输, 直到一个数据块批量传输完毕。

从时序控制方式看, 图 5-32 所示的过程是以时钟周期为单位的, 所以属于同步控制方式。时钟周期数可在一定程度上随需要而变, 如在传输过程中可插入或不插入 S_w , 因此部分引入了异步控制的策略。

在 S_1 和 S_0 状态中, 8237 并未占有总线; 从 $S_1 \sim S_4$, 8237 占有总线。所以在微机中, 一个典型的总线周期包含 4 个时钟周期。根据 CPU 的时钟频率, 可以计算出 PC 总线的总线周期基本时长, 从而计算出总线的数据传输率。

5.5.5 典型 DMA 接口举例

前面基本上是以 Intel 8237 DMA 控制器为中心, 分析它控制总线进行 DMA 传输的过程。本节将以一种微机常见的磁盘适配器为例, 讨论有关 I/O 接口方面的工作情况。选择这个实例有以下两点理由: ① 在磁盘的调用过程中, 既有 DMA 方式也有程序中断方式的具体应用; ② 磁盘适配器的组成是一种比较典型的智能型控制器结构, I/O 接口中采用了微处理器与局部存储器。

磁盘是计算机硬件系统中非常重要的外围设备之一, 通过本节的介绍, 希望读者对磁盘子系统的组成及其调用过程有深入的了解。

1. 磁盘适配器的组成

在微机中, 磁盘子系统与系统的连接方式如图 5-34 所示。系统总线通过磁盘适配器连接磁盘驱动器, 磁盘盘片在驱动器中。整个微机系统中有两级 DMA 控制器, 其中一级 DMA 控制器 8237 安装在系统板上, 作为全机的公用 DMA 控制逻辑, 管理软盘、硬盘、DMA 刷新、同步通信 4 个通道, 这一级 8237 的任务如 5.5.4 节所述。在磁盘适配器上还有一级 DMA 控制器, 这块 8237 芯片的任务是管理磁盘驱动器与适配器之间的传输。采用两级 DMA 控制器, 使适配器可以连接与控制磁盘驱动器, 并使适配器具有较大的缓冲能力, 足以协调软盘与硬盘之间的传输冲突。

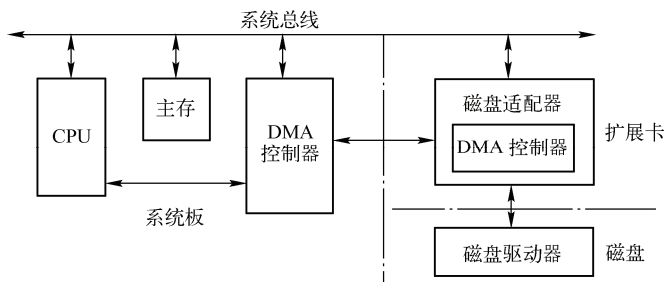


图 5-34 磁盘子系统与系统的连接方式

磁盘适配器的基本组成如图 5-35 所示, 一侧面向微机系统总线, 另一侧面向磁盘驱动器。它的组成可分为 3 部分: 一侧是面向系统总线的接口逻辑, 称为处理机接口; 另一侧是面向磁盘驱动器的接口逻辑, 称为驱动器接口; 中心是智能主控器, 包括由微处理器与局部存储器构成的一

个最小规模微计算机，以及反映设备工作特性的一组控制逻辑，还有一块 8237DMA 控制器。适配器有一组内部总线，用来连接有关部件。这种结构颇为规整，为我们设计复杂接口与局部控制器提供了一种参考模式。

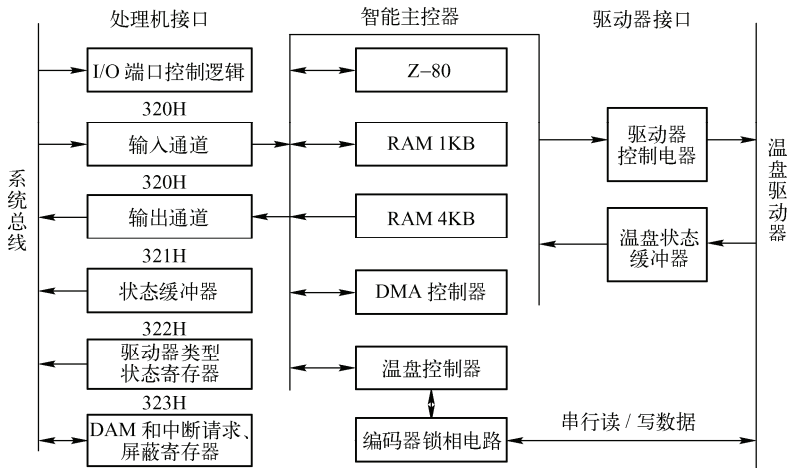


图 5-35 磁盘适配器的基本组成

(1) 处理机接口

这是与主机方面的接口逻辑。I/O 端口控制逻辑接收 CPU 发来的端口地址、读写命令，经译码产生一组选择信号，选择下述 5 种端口与相关部件。

① 输入通道。这里所指输入是对适配器而言的，由端口地址 320H 和 IOW 写命令选中，可由 74LS373（8D 锁存器/直通门）组成。通过输入通道可以输入 CPU 命令、包括磁盘寻址信息在内的有关参数、需要写入磁盘的数据等。

② 输出通道。这里所指的输出也是对适配器而言的，由端口地址 320H 和 IOR 读命令选中，可由 74LS244（8 路驱动器）组成。通过输出通道可以输出执行命令的状态，以及从磁盘读出的数据。

输入通道与输出通道占用一个端口地址 320H，辅之以读写命令区分当前选中的是输入通道还是输出通道。

③ 状态缓冲器。由端口地址 321H 选中。该缓冲器由 74LS244 组成，其中存放着 6 种状态信息：中断请求 IRQ5、DMA 请求 DREQ3、忙状态标志 BUSY、命令/数据传输命令 CMD/DATA、读/写（即 IN/OUT）、DMA 传输有效 REQUEST，可供 CPU 读取。

④ 驱动器类型状态寄存器。由 74LS244 组成，端口地址 322H 选中。早期的磁盘驱动器类型参数是由一组开关设置的，存放在本状态寄存器中，现在可由 CMOS 的 RAM 提供设置信息。这些信息包括驱动器容量、圆柱面数、磁头数等，供 CPU 进行驱动器类型检查，作为驱动器复位时的初始化参数。

⑤ DMA 和中断请求、屏蔽寄存器。它包含两个请求触发器和两个屏蔽触发器，由端口地址 323H 选中。当 CMD/DATA 为 0 时，产生 DMA 请求 DREQ3，请求传输数据字节。当 CMD/DATA 为 1（传输命令字节）且 IN/OUT 为 1（CPU 读）时，产生中断请求 IRQ5。

(2) 智能主控器

图 5-34 给出的是一种温彻斯特硬盘适配器，智能主控器是其核心，控制着磁盘存储器的操作。因此，这部分起着设备控制器的作用，其功能包括：

① 对 CPU 送来的命令进行译码,使专用的温盘控制器产生相应的控制信号,通过驱动器接口发往磁盘驱动器,驱动设备完成指定的操作。

② 通过温盘状态缓冲器检测磁盘驱动器的有关状态。

③ 通过处理机接口中的状态缓冲器,向主机报告命令的执行结果。

④ 将写入数据进行并-串转换,按 M^2F 制编码,形成写脉冲序列,送往磁盘驱动器。

⑤ 由磁盘驱动器读出数据,锁相器调整适配器振荡频率,使与读出时钟同步;通过数据、时钟分离电路,使数据信号与时钟信号相分离,从读出序列中分离出数据信号,再经串-并转换形成并行的数据字节。

⑥ 进行错误检验与纠错。

⑦ 对磁道进行格式化。

为了实现上述功能,智能主控制器由下列逻辑部件组成。

① Z-80 微处理器,它执行温盘控制程序。

② ROM,固化温盘控制程序。磁盘子系统的软件包含两级程序:一是操作系统中的磁盘驱动程序,二是适配器中的温盘控制程序,后者实现磁盘驱动器与适配器的物理操作。

③ 扇区缓冲器,由 SRAM 芯片构成,可缓存两个扇区内容,使适配器有足够的缓冲深度。

④ DMA 控制器,由一片 Intel 8237 芯片构成。因为软盘调用比硬盘频繁,微机安排软盘请求 $DREQ_2$ 比硬盘请求 $DREQ_3$ 优先级高,但硬盘读写速度(数据传输率)比软盘高 10~20 倍。为避免硬盘请求被屏蔽(优先响应软盘)带来的问题,设置了两级 DMA 传输。温盘驱动器与适配器扇区缓冲器之间的传输由适配器中的 8237 管理。扇区缓冲器与主存之间的传输,则由系统板上的 8237 管理。

智能主控制器中的 8237 芯片也有 4 个通道。其中,通道 0 用于扇区地址标志检测,一旦检测到地址标志,将产生对本 8237 的请求 DRQ_0 ;通道 1 供主控制器内部软件使用;通道 2 供专用的温盘控制器使用,当产生校验错时,提出请求信号 DRQ_2 ;通道 3 供数据传输用。

⑤ 专用温盘控制器 HDC,编码器,锁相器,数据、时钟分离电路。HDC 有专用集成芯片,控制有关读盘、写盘的信息变换。编码器可由 PROM、延迟线路、八选一驱动器等组成,需要写入的数据送入 PROM,输出其对应的 M^2F 制编码。锁相器是一种振荡频率控制电路,根据本地振荡信号与驱动器读出序列信号间的相位差,自动调整振荡频率,使其始终与读出序列保持同步。读出序列中既有时钟信号,又有数据信号,分离电路从中分离出数据信号。

写盘——从适配器扇区缓冲器 RAM 中取得写入数据,做并串转换,经编码器形成 M^2F 制代码,送往磁盘驱动器。当有一个扇区缓冲区为空时,适配器向主机提出 DMA 请求,请求主机送来写入数据。此时,适配器还有一个扇区数据可供写入驱动器。

读盘——驱动器送来串行读出信号序列,分离电路使数据信号与时钟信号分离。此时,锁相器调整本地振荡频率,始终跟踪同步于读出信号。获得的数据信号先经过串并转换,再送入扇区缓冲器暂存。

当有一个扇区缓冲区装满时,适配器向主机提出 DMA 请求,请求主机取走数据。此时,适配器还有一个扇区容量的存储空间可供存放继续读出的数据。采取这样的安排,可保证一个扇区(数据块)的连续传输,既不会在写盘过程中发生数据迟到,也不会在读盘过程中发生数据丢失。

(3) 驱动器接口

这是与磁盘驱动器方面的接口逻辑。为了使适配器连接不同型号的磁盘驱动器,一般要求双方都符合某种工业标准接口约定,早期较常使用的有 ST506 接口标准。

- ① 驱动器控制电路，用来产生对磁盘驱动器的控制信号，送往驱动器，例如：
 - ⊙ 驱动器选择 0、1、2、3 共 4 个选择信号，可选择 4 个驱动器之一。
 - ⊙ 磁头选择 20、21、22 共 3 个选择信号，经译码可选择 8 个磁头（即记录面）之一。
 - ⊙ 方向选择——控制寻道方向，为 1 则使磁头向中心方向运动。
 - ⊙ 步进脉冲——每发一个步进脉冲，驱动器中磁头在步进电动机驱动下，将移动一个道距。
 - ⊙ 写命令——为 1，写入磁盘；为 0，从磁盘中读出。
 - ⊙ 减少写电流——磁头越往内圈移动，浮动高度降低，而位密度增加。为减少内圈各位之间的干扰，应减少写电流。最外圈的写电流与最内圈的写电流相比，可相差 30%。
- ② 温盘状态缓冲器，接收驱动器状态信息，供 Z-80 判别，例如：
 - ⊙ 驱动器选中——由被选中的驱动器发回的应答信号。
 - ⊙ 准备就绪——当驱动器主轴电动机达到额定转速，且磁头重定标于 0 号磁道时，驱动器送来准备就绪信号，便可开始启动寻道操作。
 - ⊙ 寻道完成——当磁头定位于目标磁道后，驱动器送来寻道完成信号。可借此引起一次中断请求，根据从该道上读出的磁道号，判断寻道是否正确。
 - ⊙ 磁道 0——当磁头位于 0 号磁道时，驱动器送出本信号，作为磁道位置的基准。
 - ⊙ 索引脉冲——盘片每旋转一周，驱动器送出一个索引脉冲，作为磁道的开始标志。
 - ⊙ 写故障——当驱动器的写操作发生故障时，送出本信号。
- ③ 读/写数据，这是按 M²F 制的记录序列，串行传输。

2. 磁盘调用举例

磁盘调用过程涉及许多处理细节，例如：CPU 应对 DMA 控制器及磁盘适配器发出哪些命令、命令格式、发出命令的时间；在调用过程中需做哪些状态检测，以直接程序查询方式还是中断方式进行检测；一旦发现错误，适配器应提供哪些出错信息，主机又如何处理；主机为磁盘控制器设置哪些操作命令，磁盘驱动程序中包括哪些功能模块；……；不同计算机对磁盘调用的设计可能不同。限于篇幅，本节只打算粗略介绍，略去细节，以突出基本原理，因而也不完全针对某一机型。

从软件的角度看，对磁盘的调用是通过磁盘驱动程序实现的，它涉及对 DMA 控制器的初始化、对磁盘适配器的操作命令和诊断命令等。磁盘适配器和磁盘驱动器的自身操作，由适配器上的磁盘控制程序控制实现。DMA 传输由纯硬件控制实现，DMA 控制器控制系统总线、适配器与主存储器—读—写。调用过程的某些诊断，可配合程序中断予以处理。

以用 X86 构成的微机系统为例，其系统软件中有一个基础模块 BIOS（基本输入/输出系统），常固化在主机板的 ROM 中。磁盘驱动程序就是 BIOS 中的重要组成之一。

操作系统的温盘驱动程序以软中断 INT 13H 调用，它包含一个主程序框架和 21 个功能子程序模块，可向磁盘控制器发出 22 种操作命令与诊断命令。

主程序的功能包括测试驱动器参数判别可否调用；设置命令控制块，其中给出圆柱面号、磁头号、扇区号、传输扇区个数、寻道的步进速率、功能子程序模块号；转入某个功能子程序；在传输完毕后判断调用是否成功等。

共有 21 个功能子程序模块可供选用，例如：磁盘复位；读取磁盘操作状态；读盘，即将指定扇区内容读入主存；写盘，即从主存写入指定扇区；检验指定的扇区；格式化指定的磁道，设置故障扇区的标志；从指定的磁道开始格式化；返回当前驱动器参数；初始化驱动器性能参数；长读（每扇区 512 字节+4 校验字节）；长写（每扇区 512 字节+4 校验字节）；磁道寻找；磁盘复

位 (DL 寄存内容为 80H~87H); 读扇区缓冲器; 写扇区缓冲器; 测试驱动器准备状态; 诊断磁盘控制器中的 RAM; 诊断驱动器; 控制器内部诊断。

进入功能子程序后, 以主程序设置的命令控制块为基础, 填入有关控制信息后形成设备控制块, 发往磁盘适配器。设备控制块 1~6 字节不等, 按照约定格式形成 22 条硬盘控制器 (HDC) 命令, 例如: 测试驱动器就绪; 重新校准; 请求检测状态; 格式化驱动器; 就绪检验; 格式化磁道; 格式化坏磁道; 读; 写; 寻道; 预置驱动器特性参数; 读 ECC 猝发错长度; 从扇区缓存读出数据; 向扇区缓存写入数据; RAM 诊断; 驱动器诊断; 控制器内部诊断; 长读; 长写等, 还有几条保留未定义。这些 HDC 命令体现了各功能子程序的主要功能。

磁盘适配器中的微处理器, 执行磁盘控制程序, 根据上述命令信息进行相应的操作。每当执行 HDC 命令结束时, 都要向主机回答一个表示完工的状态字节。该字节通知主机, 在执行该命令期间是否出现错误。如果允许中断, 状态字节的传输与处理可以采用中断方式。当磁盘适配器准备好传输状态字节时, 向主机发出一个中断请求 IRQ5, 主机通过中断处理程序取走状态字, 从而结束一个 HDC 命令, 适配器清除“忙”标志。

如果操作中出现错误, 适配器形成 4 字节的检测数据。其中, 字节 0 给出错误类型及错误代码, 字节 1~3 则指明驱动器号、圆柱面号、磁头号、扇区号。错误类型及错误代码指出 4 类 27 种错误之一, 例如: 磁盘控制器没有检测到来自驱动器的索引信号; 寻道之后没有收到寻道完成信号; 检测到驱动器出现写故障; 选择驱动器后, 没有收到就绪信号; 没有 0 道信号; 寻道不能结束; 标志区读出校验错; 数据读出错; 未能检测到磁道地址标志; 未能找到目标扇区; 寻道错; 坏磁道; 适配器收到的是无效命令; 非法磁盘地址 (超出最大范围); RAM 错 (在 RAM 扇区缓冲器的诊断中发现数据错); 在控制器内部诊断中发现其程序存储器的检查和有错; 在控制器内部诊断中发现 ECC 多项式错; ……; 当主机从状态字中发现有错时, 可进一步从适配器取回反映出错状态的检测数据, 以判明错误性质。

当主机执行完一个功能子程序后, CPU 也将形成出错与否的信息代码。进位标志 CF 为 0, 表示操作成功, 此时寄存器 AH 内容为 0, 表示没有错误; 可结束磁盘调用, 返回用户程序的主程序。如果进位标志 CF 为 1, 表示操作有错误发生; 此时 AH 中给出错误码, 指出错误性质, 如磁盘 I/O 命令错误、地址标志未找到、需要的扇区未找到、复位故障、驱动参数有问题、DMA 超越 64K 范围、坏磁道、读盘出现校验错、ECC 校正数据错误、磁盘控制器故障、寻道故障、设备未响应、发现未定义的错误、检测操作出现故障等。发现有错后, 或重新执行, 看是否属于干扰造成的偶然性故障, 或通过显示器报告故障信息。

下面以 X86 机读/写磁盘为例, 说明磁盘调用的大致过程, 其间略去了一些细节。

(1) 以软中断“INT 13H”调用温盘驱动程序, 并在寄存器 AH 中写入所需功能子程序号: 读盘, AH=02H; 写盘, AH=03H。

(2) 在温盘驱动程序的主程序段中, 设置命令控制块, 其中给出磁头号、圆柱面号、起始扇区号、扇区数、寻道步进速率。

(3) 根据 AH 值, 转入相应功能子程序。

(4) 在读/写盘子程序中, 进行 DMA 初始化。

向 DMA 控制器 8237 送出方式控制字, 其中包括 DMA 传输方向、传输方式 (单字节方式或数据块连续传输方式)、是否选择自动预置方式、地址增/减方式。初始化温盘占用的 DMA 通道, 即向 8237 送出主存缓冲区首址、交换字节数。判断 DMA 传输量是否超过 64 KB。若越界, 做出错处理; 若正常, 向磁盘适配器 (端口 323H) 送入允许信息, 允许 DMA 请求和中断请求。

(5) 在读/写子程序中检测适配器状态(端口 321H), 然后以主程序设置的控制块为基础形成设备控制块, 发往适配器端口 320H, 产生 HDC 命令, 启动寻道和读/写操作。

(6) 当寻道完成时, 磁盘驱动器向适配器发出“寻道完成”信号, 适配器判别寻道是否正确。若寻道正确, 可启动读/写操作。若不正确, 可重新定标, 让磁头回到 0 道, 然后重新寻道。若仍不正确, 则产生寻道故障信息。

(7) 当磁头找到起始扇区时, 开始连续地读/写, 将读出数据送入适配器的扇区缓冲器, 或将扇区缓冲器中的数据写入磁盘扇区。

(8) 当适配器准备好 DMA 传输时, 适配器向 8237 提出 DMA 请求。

读盘: 当磁盘适配器的扇区缓冲器有一个扇区缓冲区装满时, 提出 DMA 请求。

写盘: 当扇区缓冲器有一个扇区缓冲区为空时, 提出 DMA 请求。

(9) DMA 控制器申请并接管系统总线, 实现 DMA 传输, 并相应修改主存地址、传输字节数。

(10) 当批量传输完毕, DMA 控制器发出结束信号 \overline{EOP} , 终止 DMA 传输。适配器再向主机提出中断请求。

(11) 主机的读/写盘子程序在接到中断请求 IRQ_5 后, 从适配器取回完工状态字节, 判断 DMA 传输是否成功。若有错, 取 4 个检测数据字节, 进行出错处理。若成功, 向适配器送出屏蔽请求的屏蔽字, 然后返回, 结束调用。

在上面的描述中, 有些内容是与具体计算机调用方式有关的, 如: 以何种方式调用磁盘驱动程序, 命令格式与传输, 做哪些检测等, 有些内容是具有共性的, 如: DMA 控制器初始化、向适配器送出寻址信息与传输方向, 以纯硬件方式实现 DMA 传输, DMA 控制器控制传输过程的终止, 以中断方式进行结束处理等。

5.6 IOP 和 PPU

现代计算机系统中通常会连接许多种类各异的输入输出设备, 可能既有高速设备也有低速设备, 甚至同一类设备也可能会有多台。在这种情况下, 每个外设都配置一个专用的 DMA 控制器不太现实, 且多个控制器并行工作还会造成存储器访问的冲突, 反而会降低系统的性能。解决多个设备 DMA 问题的基本思路就是设置一个可被多种外设共享的专用控制器, 在此思路发展出了 IOP (Input/Output Processor) 的概念。IOP 是一种设置专用处理器来控制具体 I/O 操作的数据输入/输出控制方式, 它把主机 CPU 从繁杂的 I/O 控制中解放出来, 实现了 I/O 控制与 CPU 的并行性, 显著提升了计算机系统的 I/O 性能。

通道 (Channel) 是一种最典型的 IOP 方式, 本节将重点介绍通道的工作原理。

5.6.1 通道的系统结构

通道是一个带有专用处理器的高级输入输出控制部件, 其基本任务是通过执行程序来管理主机与外设之间的数据输入和输出。通道控制器有自己的指令, 即通道指令, 能够通过程序控制多个外设, 还能提供 DMA 共享功能。一般在大中型计算机中都采用了通道, 采用了通道的大型计算机系统基本结构如图 5-36 所示。从逻辑结构层次上看, 采用了通道后, 会涉及三个层面的连接交互, 即“主机 - 通道”层面、“通道 - 接口”层面、“接口 - 外设”层面。通道与外设之间的接口是计算机硬件系统的一个重要界面, 为了便于用户配置设备, “通道 - 接口”层面一般采用 I/O 总线, 使各外设与通道之间有相同的接口线和工作方式, 在更换设备时, 通道就不需要任何改变。

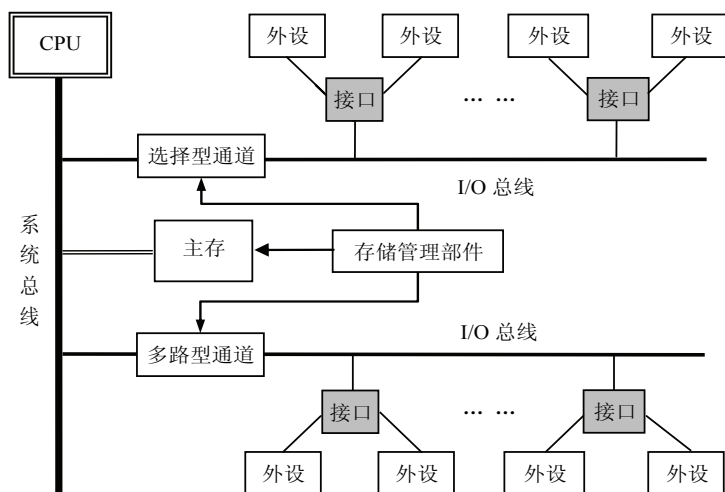


图 5-36 带通道的大型计算机系统结构

当需要进行主机与外设之间的数据输入输出时，CPU 先启动通道，并将“数据传输”的具体控制任务交给通道，通道负责控制主机与接口之间的具体 I/O 操作（多数情况下采用的是 DMA 方式），CPU 只负责“数据处理”功能。这样，通道 I/O 系统和 CPU 就能分时使用主存，也能实现 CPU 与 I/O 系统的并行工作。

5.6.2 通道的类型

一台计算机可以配置多个通道，一个通道上有可以挂接多个外设。根据数据传输方式的不同，通道总体上可以分为两种：选择型通道和多路型通道。

（1）选择型（Selector）通道

选择型通道（只能以突发模式工作）可连接多个外设，但这些设备不能同时工作，在某个时间段内只能选通 1 个设备进行工作，只有当这个设备的全部通道程序执行完后，才能选择其他设备进行工作。选择通道主要用于连接高速的外部设备，例如磁盘等。

（2）多路型（Multiplexer）通道

多路型通道也可以连接多个外设，兼容选择型通道，在某一时段内可以同时选通多路设备进行工作，且允许这些外设按一定的单位轮流进行数据的输入、输出。根据每次数据传输单位的差异，多路型通道又分为字节多路型通道和数组多路型通道。

① 字节多路通道。字节多路通道是一种简单的分时共享通道，主要用于连接多路低速或中速设备。通道选择一路设备后，该设备开始传输数据，但只能传输 1 字节，若该设备有多字节数据需要传输，则该外设需要多次请求，被通道多次选择才能完成传输。

字节多路型将通道的可用传输时间分为若干个时间片，每个时间片传输 1 字节，由多个外设分时方式共享一个通道。字节多路型通道主要用于连接多种低速外设。

② 数组多路通道。数组多路型通道在物理上可连接多路高速外设，数据以成组（块）交叉的方式进行传输。通常，一个外设在进行工作时，除了数据传输，还包括寻址等操作。数组多路通道的基本思想是：当某个设备进行数据传输时，通道只为该设备服务；当设备在执行寻址等非传输型操作时，通道暂时断开与这个设备的连接，挂起该设备的通道程序，去执行其他设备的通道程序，为该设备的输入输出服务。

选择型通道和多路型通道的性能对比如表 5-3 所示。

表 5-3 选择型通道和多路型通道的性能对比

| 性能 | 通道类型 | 选择型 | 多路型 | |
|---------|------|-----------|----------|---------|
| | | | 字节多路 | 数组多路 |
| 单次数据传输量 | | 不定长, 全部数据 | 1 字节 | 定长数据块 |
| 适用范围 | | 高优先级的高速设备 | 大量的低速设备 | 大量的高速设备 |
| 工作方式 | | 独占通道 | 各设备按字节交叉 | 各设备成组交叉 |
| 通道的共享性 | | 独占, 完成后释放 | 分时共享 | 分时共享 |
| 选择设备的次数 | | 仅 1 次 | 可能多次 | 可能多次 |

例如, 假设有 P 台设备连接在通道上, 每台设备传输数据需要经历设备选择和数据传输 2 个时间段。分别用 TS_i 、 TD_i 和 n_i (字节) 来表示第 i 台设备的选择时间、传输 1 字节数据所需时间和数据传输量, 其中 $i=1\sim P$ 。

对于选择型通道, 完成 P 路设备的数据传输所需的总时间为:

$$T_{\text{select}} = \sum_{i=1}^P (TS_i + TD_i \times n_i) \quad (5-2)$$

对于字节多路型通道, 完成 P 路设备数据传输所需的总时间为:

$$T_{\text{byte}} = \sum_{i=1}^P (TS_i + TD_i) \times n_i \quad (5-3)$$

对于数组多路型通道, 假设每一次传输的数据量为 m_i 字节, 则完成 n_i 字节数据共需要进行 n_i/m_i 次断点续传才能完成, 所以 P 路设备的数据传输所需的总时间为:

$$T_{\text{block}} = \sum_{i=1}^P (TS_i + TD_i \times m_i) \times \frac{n_i}{m_i} \quad (5-4)$$

从式 (5-2) ~ 式 (5-4) 可以看出, 在传输数据量相同的情况下, 选择型通道花费在设备选择上的时间开销最少, 传输效率最高, 但其最大缺点是当一个设备被通道选择使用后, 要等它的数据全部传输完毕才会被通道释放, 因此单次通道占用时间比较长, 这会导致通道对其他外部设备传输请求的响应时间延迟会很长。

字节多路型通道与选择型通道刚好相反, 因为要多次选择设备, 故其花费在设备选择上的时间中开销最多, 传输效率也最低。由于被选择的设备一次只能传输 1 字节数据就要被通道释放, 故单次通道占用时间比较短, 因此通道对其他外设传输请求的响应时间延迟最短。

数组多路型通道的数据传输效率和传输请求响应延时介于选择型通道和字节多路型通道之间, 实际是这两种通道的一种优化折中, 在系统效率和响应时间上取得了完美平衡。

5.6.3 通道的工作原理

通道一般由几个基本的部件组成, 主要包括: 通道控制器、通道状态寄存器、中断电路、通道地址字寄存器和通道命令字寄存器及 I/O 缓冲等, 结构如图 5-37 所示。

在介绍通道的工作原理之前, 先介绍通道中的几个主要部件和相关术语。

通道控制器: 通道内部设置的一个专用控制器, 用来产生控制通道操作的各种信号, 从功能和作用上看, 非常类似于 CPU 中的微命令发生器。

中断控制逻辑: 用来产生并向 CPU 发送中断请求信号的逻辑电路单元。两种情况下通道会产生中断请求: 一种是数据传输正常结束; 另一种是传输过程中发生某种异常。

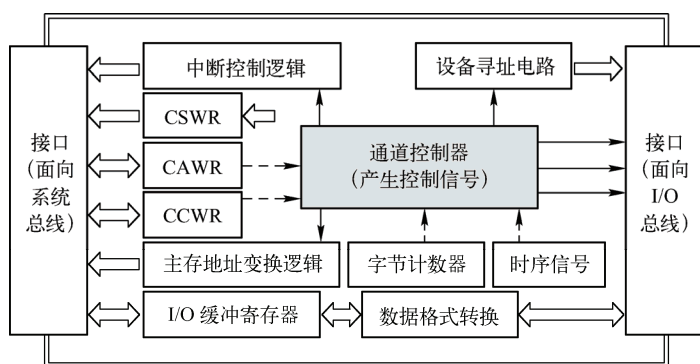


图 5-37 通道内部的逻辑结构

I/O 指令：属于访管指令（涉及操作系统的 I/O 管理程序），是主机 CPU 专门用来启动、停止通道，测试通道和外设状态等的简单指令，不直接参与控制具体的 I/O 操作。

CSW（Channel Status Word，通道状态字）：类似 CPU 的 PSW（程序状态字），主要用于记录 I/O 操作结束的原因及操作结束时通道自身和外设的状态。CSW 通常存储在固定的主存单元中，以便 CPU 能快速地读取所需的各种状态信息。

CSWR（Channel Status Word Register）：即通道状态字寄存器，专门用来暂存 CSW 各部分信息的寄存器。

CAW（Channel Address Word，通道地址字）：用于指明通道程序的首地址，通常保存在一个固定的内存单元中。

CAWR（Channel Address Word Register）：即通道地址字寄存器，专门用来存放从主存中读取到的 CAW 信息。

CCW（Channel Command Word，通道命令字）：即通道指令，是通道控制器产生各种控制信号来控制 I/O 操作的主要依据，类似于 CPU 执行的机器指令，由命令码、主存地址、标志码及传输字节数等代码段构成。

CCWR（Channel Command Word Register）：即通道命令字寄存器，它是一种用来暂存 CCW 的专用寄存器。

通道程序：由一条或若干条 CCW 组成，也常称为通道指令链，由通道取指机构从主存中逐条读取并暂存在 CCWR 中。通道程序一般由 CPU 根据 I/O 请求来进行编制，再由 I/O 管理程序将其存放在主存中，并将程序在主存中的首地址写入到 CAW 中。

通道的基本工作过程如图 5-38 所示，从主机和通道角度来描述：

- （1）用户程序提出 I/O 请求。
- （2）CPU 响应 I/O 请求，切换至管态，执行 I/O 管理程序，组织通道程序并将其保存在内存中并记录通道地址字，安排主存区，确定主存缓冲区首地址和数据传输量等。
- （3）初始化通道号和设备号，向通道发出启动命令，CPU 返回用户程序执行。
- （4）通道接收到启动命令后，从固定的主存单元中读取 CAW 并暂存到 CAWR。
- （5）通道暂存设备号，然后对外设进行寻址，连接到目标外设。
- （6）根据 CAW 读取通道程序的首条 CCW 并存入 CCWR 中，然后启动外设。
- （7）通道继续逐条读取并执行当前通道程序剩余的 CCW 来控制具体的 I/O 操作，直到最后一条 CCW 执行完毕。在此过程中，更新 CSWR 的内容。
- （8）通道控制的 I/O 操作结束后，或者在数据传输过程中发生了某种异常，通道向外设发出

结束命令，向 CPU 发出中断请求，将 CSWR 的内容保存到固定的主存单元。

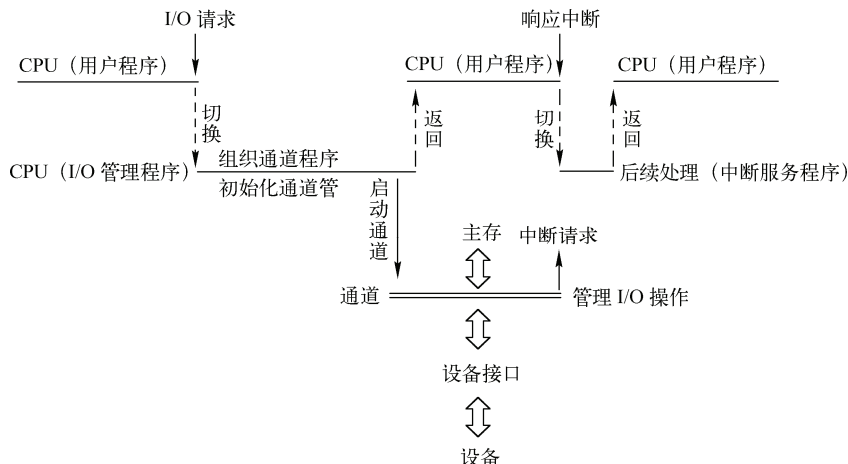


图 5-38 通道的工作过程示意图

(9) CPU 暂停执行当前用户程序, 响应通道的中断请求, 然后切换到通道的中断服务程序执行。在中断服务程序中, 通过主存中存放的 CSW, 了解本次传输的情况, 并进行 I/O 善后处理, 如输入/输出异常处理和数据校验等。

从通道的工作过程可以看出,主存-设备之间的数据输入/输出操作是在通道的控制下完成的,而通道对输入/输出的控制是通过执行通道程序来实现的,此外通道又是被主机指挥和控制的。具体的输入/输出过程不需要主机干预,主机 CPU 可以和通道并行工作,主机仅在管理通道(如启动或停止通道)、I/O 操作发生异常或数据输入/输出结束时才参与处理。

从总体上来看，通道最核心的功能就是执行通道程序，控制主存与外部设备之间的数据 I/O 操作，因此通道的基本功能可总结如下：

- ① 接收 CPU 发出的 I/O 指令，外设寻址，控制外设。
- ② 执行通道程序，向外设发送 I/O 操作控制命令。
- ③ 组织和控制主存与外设之间具体的数据 I/O 操作，如数据缓冲、主存寻址、数据量控制和数据格式的转换等。
- ④ 记录状态信息并更新 CSWR，并将 CSW 保存到固定的内存单元，供 CPU 使用。
- ⑤ 向主机 CPU 发出 I/O 中断请求。

此外，设备控制器的完成的主要功能也可概括为：

- ① 从接口接收通道的 I/O 控制命令,如启动、停止等,以及向外设发送各种控制信号控制外设完成指定的 I/O 操作。
- ② 通过接口向通道提供外设的状态信息。
- ③ 按通道和接口的标准,对不同的外设非标信号进行格式转换。

在使用通道的这种 I/O 方式中,除了主机系统的处理器外,一般采用了另外一个可以执行通道指令来进行具体 I/O 控制的处理器,见图 5-37,因此通道是一种典型的 IOP 方式。同时,该专用于 I/O 控制的处理器一般位于主机之外,因此有的技术资料也把这个处理器称为外围处理器(Peripheral Processor, PP)。

IOP 独立于主 CPU 进行具体的 I/O 操作控制,但它也要接受主 CPU 的控制,因此逻辑上它仍然属于主机硬件系统的大范畴。有些 IOP 还能提供数据的变换、搜索、字装配/拆卸能力,如

从输入/输出方式上看,除 IOP 方式外还存在 PPU (Peripheral Processor Unit, 外围处理机) 方式。PPU 一般独立于主机系统,甚至有自己独立的、完善的指令系统,能够完成算术/逻辑运算、读/写主存以及控制外设 I/O 操作等。从这一点看,PPU 与一个完整的计算机系统并无差异,因此有的场合甚至直接用通用计算机作为 PPU。一般在大型的计算系统中会采用 PPU 来构建多机系统,从而使其具备令人惊讶的高效计算和 I/O 控制能力。PPU 已超出了一般 I/O 子系统的概念,本书只对其做简要介绍,不做深入讨论。

| | | | | | | |
|------|-------|---------|-------|------|----------|------|
| 总线 | 总线宽度 | 总线带宽 | 主设备 | 从设备 | 直接程序传输方式 | 中断 |
| 软中断 | 硬件中断 | 可屏蔽中断 | 非屏蔽中断 | 向量中断 | 非向量中断 | 中断向量 |
| 向量地址 | 中断向量表 | 中断隐指令操作 | DMA | | | |

340

第6章 输入/输出设备及接口

中央处理器（CPU）、主存储器（Main Memory, MM）构成计算机的主体，称为主机。主机以外的大部分硬件设备被称为外围设备（或输入/输出设备，I/O 设备）。输入/输出（I/O）接口是主机与输入/输出设备之间的交接面，通过它才能实现主机与输入/输出设备之间的信息交互。

输入/输出设备是计算机系统与人或其他设备、系统之间进行信息交换的装置。人们常用数字、字符、文字、图形、声音等形式来表示各种信息，而计算机所能处理的是以电信号形式表示的数字代码（实质上就是相应的一系列 0、1 表示的二进制代码）。因此，需由输入设备将各种原始信息转换为计算机能识别处理的信息形式，并输入计算机；由输出设备将计算机处理的结果转换为人或系统能识别的信息形式，并对外输出。

这里所定义的输入/输出，是以主机作为基准点，送入主机称为输入，由主机送往外部称为输出。就其与主机的关系而言，输入设备与输出设备基本上是相同的，只是数据传输方向不同而已。有些设备既是输入设备，也是输出设备（如磁盘）。因此，常将这两种设备统称为输入/输出设备。

在整个计算机系统中，一般用户的外围设备占总成本的 50% 以上，从故障率来看，约有 80% 的故障发生在输入/输出子系统上。外围设备不仅种类繁多，它们的构成和工作原理也差别甚大，且与主机的连接方式复杂多变。因此，本章在按功能分类予以概述之后，重点介绍几种最常用的基本 I/O 设备：键盘、显示设备、打印设备、音视频设备和条码设备等。

6.1 概述

6.1.1 输入/输出设备的一般功能

用户在使用计算机系统时，接触最多的是输入/输出设备。输入/输出设备是计算机主机与外界实现联系的装置，对计算机系统使用方便与否影响很大。输入/输出设备的种类繁多，不胜枚举。但总的来说，输入/输出设备具有以下几个功能。

1. 完成信息的转换

通常，人们习惯用字符、声音、图形、图像等来表达信息的含义，处理机只能识别和处理用“0”和“1”表示的二进制代码。因此，在计算机进行数据处理时，首先必须将处理程序、原始数据及操作命令等信息变成处理机能识别的二进制代码；同样，处理机处理的结果要告诉用户，也必须变换成为人们熟悉的表示形式。这种处理机与外界联系时信息形式的变换通过输入/输出设备才能实现。

2. 实现人机交互

无论计算机用于何处，都要由人去操作，尽管在自动控制或某些其他领域里，人与计算机可能不直接接触，但在研制、开发程序的过程中，人仍然需要直接与计算机交互联系。实现这一联系的装置仍然是输入/输出设备。例如，有了显示器、键盘、鼠标等外部设备，用户可直接、方便地与计算机实现多种形式的交互，这样可以大大提高计算机的处理效率，加速计算机的应用推广，并充分发挥人的智能作用、提升计算机的用户体验。

3. 存储信息资源

随着计算机功能的增强，系统软件的规模和被处理的信息量也日益扩大，因此不可能将它们全部存于主存中。在这种情况下，需要有一种能扮演系统软件和各种数据信息驻留地的输入/输出设备，如硬盘和 U 盘。

4. 促进计算机应用领域的拓展

输入/输出设备是计算机在各领域应用的重要物质基础，早期的计算机主要用于数值计算，输入/输出设备比较简单。随着计算机应用范围的扩大，很快超出了数值计算的范围，输入/输出设备作为计算机系统的重要组成部分，便以多种多样的形式进入各领域。“模/数”(A/D)、“数/模”(D/A)转换装置的出现，使计算机适应于工业自动化的需要。图形数字化仪、智能式绘图仪及带光笔的交互式字符图形显示器为计算机在辅助设计方面的应用提供了有力的支持。语音输入识别装置、传真机、图像输入设备的出现，使办公自动化深受人们的重视。此外，计算机在商业、金融、情报等部门的应用，出现了磁卡和条形码阅读机。在医疗部门，已采用计算机断层扫描设备获取人体内部清晰的图像。总之，必须有适用的输入/输出设备，才能使计算机在各个领域获得广泛的应用。

6.1.2 输入/输出设备的类型

为了增强系统功能，计算机系统配置的外围设备越来越多。由于计算机极其广泛地应用在各种领域，除了配置一些基本的输入/输出设备外，还可能配置一些与应用领域相关的外围设备。这些设备的工作原理，除常见的机电式、电子式外，还涉及各种新的技术成果，涉及各式各样的物理、化学机制，而且在不断发展中。外围设备的一个显著特点就是多样性，这也导致了多种分类，如按功能与用途、工作原理、速度快慢、传输格式等进行分类。如果不是从设备研制本身，而是从计算机系统组成的角度分类，一般选择按设备在系统中的作用来划分，可将它们分为 6 类。同一种设备可能具有其中几种功能。

1. 输入设备

如前所述，输入设备将外部的信息输入主机，通常将操作者（或广义的应用环境）提供的原始信息转换为计算机能识别的信息，然后送入主机。例如，将符号形式（如字符、数字等）或非符号形式（如图形、图像、声音等）的输入信息转换成代码形式的电信号。常见的输入设备有键盘、穿孔输入设备、数据站（脱机录入装置）、图形数字化仪、字符输入与识别装置、语音输入与识别装置，光笔、鼠标、跟踪球、操纵杆等辅助装置。

键盘是目前最常用也是最基本的输入装置。通过键盘可输入程序和数据（字符方式），甚至直接利用键盘编程或编制文件。键盘操作也是主要的人机界面之一，在人机对话中，通过键盘向计算机发出命令，或提供回答信息。

早期曾广泛使用穿孔信息载体，如穿孔纸带、穿孔卡片。每个孔的位置记录一位二进制信息，有孔为 1，无孔为 0。相应地，在输入设备中有纸带输入机（光电式、电容式）和卡片输入机。穿孔纸带的使用源于邮电通信，纸带中的一行穿孔信息表示一个字符，分为五单位（每行 5 位）和八单位（每行 8 位）两种；通过光电检测或电容检测进行识别，并转换为相应的电信号。在通用计算机中，不再使用穿孔纸带，但在数控机床一类设备中，目前还作为一种可供选用的方式。

穿孔卡片输入机使用纸质穿孔卡片，每张卡片按行、列记录穿孔信息，并规定其编码含义，每张卡片可记录 80 个字符。与纸带相比，卡片容易组织，便于插入或删除，曾作为 IBM360 系统

的基本输入设备，但现在早已被淘汰了。

数据站是一种数据录入装置。为了不让数据录入占用主机宝贵的运行时间，大批量数据录入往往采取脱机录入方式。即先在专门的录入装置（有的就称为数据站）中由人工录入数据（装置一般具有显示、编辑、存盘/带功能），录入结果存入软盘或磁带中，再联机送入主机。

图形数字化仪将图形、图像信息转变为数字代码，然后送入计算机处理。最常见的方法是通过自动光扫描或运用摄像机，将图形、图像信息以像素为单位转换为数字信息。现在广泛应用的扫描仪，从简易的手持式扫描仪到复杂的自动扫描仪、摄像扫描仪，种类很多。对于比较简单、较规整的图形，也可以用人工移动光标的方法跟踪已有图形，或绘制图形。例如，在 CRT 显示器屏幕上，用光笔移动光标，实现跟踪定位和绘制。又如，在图形输入板上，用画笔在输入板上跟踪图形移动，将图形的位置坐标（如一根直线的起点与终点）转换为数字，输入计算机，所采用的原理有电磁感应式、超声波式等。

随着模式识别技术的发展，出现了智能输入设备，让计算机直接“会看、会听”的梦想已在很大程度上实现。如光学字符阅读机，在光扫描时，字符在纸面上各点的反射光强度大大低于空白点的反射光强度，据此转换为电信号送入计算机。又如，磁性字符阅读机，字符是用磁性墨水书写的，通过磁头对其扫描，也可转换为电信号。在计算机中运用模式识别技术可判断字符是什么，从而产生相应的字符编码。对规整的印刷体字符，自动识别率已经很高。对人工书写体字符，计算机经过学习，也能达到很高的识别率。

语音输入及处理技术的发展很快。将声波转换为数字信号，或者从中提取某些特征参数，然后输入计算机，根据需要进行编辑处理、还原，这方面的技术已较成熟，广泛用于多媒体系统中。但语音信号远比字符信息的模式复杂，因此语音识别技术还未成熟。目前，只能对有限的语音信息进行学习、识别，如对有限操作人员的操作命令（口语），经过学习后可以自动识别和执行。

2. 输出设备

如前所述，输出设备将计算机处理结果输出到外部，通常是将处理结果从数字代码形式转换成人或其他系统能识别的信息形式。例如，显示器或打印机提供人能识别和理解的信息、程序执行的结果、运行状态，或者人机对话中计算机发出的询问、提示等。又如，计算机将结果输出到磁盘、磁带，下次可再输入计算机（本身或其他计算机）进行处理，也可作为其他系统所能识别的信息。常见的输出设备有显示器、打印机、绘图仪、复印机、电传机等办公设备，以及语音输出装置和早期的穿孔输出设备（纸带穿孔机、卡片穿孔机）等。

显示器和打印机都是属于基本配置的重要输出设备，后面将介绍。打印结果一般可以长期保存，因此打印机被称为硬拷贝设备。显示结果不能保存，因此显示器常常被称为软拷贝设备。

绘图仪是常用的图形输出装置，按工作原理可分为多种。常见的绘图仪是用墨水绘笔绘制图形的，绘笔由步进电动机驱动；一个步进电动机使绘笔沿 X 轴方向运动，另一个步进电动机使绘笔沿 Y 轴方向运动。笔架可使绘笔抬起（离开纸面），或接触纸面。彩色绘图仪有数支不同颜色的绘笔，可根据需要选取。这种绘图仪称为平板式绘图仪，也有滚筒式绘图仪。还有一种光绘仪直接使底片按绘制图形感光，可用于制作印制电路板。绘图仪由计算机控制，按计算机输出的代码进行绘制。

办公室自动化是计算机应用领域之一，现代办公设备也就成了计算机系统的输入/输出设备。计算机连接电传机，成了办公室与外部的通信设备。电传机收到报文后可以直接输入计算机，自动识别处理。计算机形成报文后，可以直接送往电传机，向外发送。语音输出得到广泛应用，如自动广播系统、警报系统、计算机自身的语音提示等。产生语音的方法可分为两大类。一类是语

音合成，计算机内存中有各种基本音素的数字化信息，可以根据需要自动合成各种词汇和语句。这种方法的优点是可生成的词汇数几乎不受限制，缺点是语音质量尚差，与自然的语音尚有明显差别（即自然度较差）。另一类是将标准语音数字化后输入计算机，以常用词汇或常用语句为单位进行组织、存储，构成语音库。当需要语音输出时，从语音库中读取所需的词汇和基本语句，编辑为所需的语句，向外播出。这种方法产生的语音自然度较高，但由于存储容量的限制，语音库能存储的词汇有限，一般用于专用领域。

3. 外存储器

外存储器是指主机之外的一些存储器，如磁盘、磁带、光盘、磁泡等。它们既是存储器子系统的一部分，也是一种输入/输出设备，既是输入设备也是输出设备。外存储器的任务是存储或读取数字代码形式的信息，一般不担负信息转换，所以常将它们视为输入/输出设备中专门的一类。

4. 过程控制设备

当计算机对一个生产过程或实验过程实施实时控制时，必须从被控对象中取得各种参数，如位移、转角、压力、速度和温度等，它们是模拟量，通过各种类型的传感器，可将相应的非电量转换为电信号。在许多情况下，传感器的原始输出是模拟信号，如以电流或电压的大小反映被测物理量；经过模数（A/D）转换，将模拟信号转换为二进制数字信号，才能送入计算机处理。实现调节控制的各类执行元件，如电磁阀、电动机等，有的可由数字信号直接驱动，有的需由模拟信号驱动。对于后一种情况，则需要将计算机输出的模拟信号，经数模（D/A）转换，转换为模拟信号，再进行功率放大，才能驱动执行元件。这就需要 A/D 或 D/A 转换设备。A/D、D/A 设备均属于过程控制设备，有关的检测设备也属于过程控制设备。

5. 数据终端设备

与计算机信息网络的一端相连接的设备又称为终端设备。可以通过终端设备在一定距离之外操作计算机，以及输入信息、获得处理结果。

在不同领域的习惯语中，“终端”一词的含义可能不同。例如，在谈论一套计算机系统带有多少终端时，一般是指可供多个用户同时使用的分时终端数，一个单用户方式的个人计算机称为单终端。一个可以同时接纳 32 个用户上机的分时系统，带有 32 台终端，有的系统将这些分时终端称为工作站（与现在流行的使用高档微机组成的工程工作站 EWS 不同）。每个用户至少需要一个键盘和一个显示器，这样的终端从构成的角度出发叫做键盘显示终端。

在计算机信息网络中，“终端”一词的含义更侧重于与计算机有一定距离，需由通信线路连接，即在计算机通信线路的另一端的设备，如键盘显示终端、打印终端、电传终端或其他通信终端等。按与计算机之间的距离，终端可分为本地终端和远程终端两类。

与主机距离较近的终端称为本地终端，如前述分时系统的多个终端，一般在同一个机房中。又如，在计算机局域网络中，连接距离有限，往往在同一座大楼、厂房之中，所连接的终端也属于本地终端。与主机距离较远的终端称为远程终端。在主机与远程终端之间往往需要通过交换装置（如交换机、调制解调器等）与光纤或公共电话线相连接。计算机输出的数字代码信号，经交换器，能在光纤、公共电话线上或其他传输介质上远距离传输高频数字、模拟信号，然后送往远程终端或系统。

6. 数据通信设备

为了实现全球信息资源的共享以及异地之间的通信，须用专用的设备把计算机连接成网络，

这就需要数据通信设备。

6.1.3 输入/输出设备与主机之间的信息交换

I/O 设备与主机之间的关系归根结底是信息的输入或输出，泛称为信息交换。因此，我们关心信息交换中所使用的代码格式、传输格式（串行、并行）和传输速率。

1. 代码格式

虽然可能传输的信息有许多种，从数值型到非数值型，从字符型到非字符型等，但在进入计算机或由计算机输出时，一般采用二进制编码形式，计算机才便于识别处理。即使是非数值型、非字符型的信号，如图像信息，也常常在数字化后，被转换为一串数字编码进行传输。为了使所传输的信息格式具有通用性，能为各种计算机所识别，国际上普遍采用 ASCII 码作为标准的代码格式。有些大公司除了支持 ASCII 码外，也有自己的专用信息交换码。

2. 传输格式

主机与外围设备间的数据传输格式有以下两种。

(1) 并行传输

同时用多根传输线，并行地传输一字节或一个字，称为并行传输。大多数外围设备适用于以字节为单位传输数据，因为一个字符的 ASCII 码为 7 位，加上 1 位奇偶校验位，正好是 1 字节。计算机系统则取决于系统总线的数据位数（数据通路宽度）。在低档微型计算机 PC-XT 中，系统总线的数据位数也是 8 位，正好与设备相同，连接比较方便。为了提高计算机的传输速率，系统总线的数据位数常设计得较多。如微型计算机经历 16 位总线发展到 32 位总线、64 位总线。因此，在外围设备与系统总线之间需要接口部件，实现字节与字之间的组装和拆卸。

采用并行传输格式，虽然数据的传输速率高，但相应的硬件代价也高，因此这种方式一般用于传输距离近且速率要求高的设备。

(2) 串行传输

采用单根信号传输线（对公共地形成电位差），或采用一对传输线（一根信号线，一根地线），逐位地串行传输数据代码，称为串行传输。这种方式的硬件代价较低，但传输速率较低，一般用于允许低速传输或远距离传输场合。例如，人工按键的操作速率以数分之一秒计，允许在键码传输过程的某些环节采用串行发送。又如打印机，机电型的打印操作速度也较慢，因此有的打印机采取串行接口传输数据。注意，远距离通信线路代价比较昂贵，一般采用串行传输。所以，在计算机与调制解调器之间一般采用串行传输。

既然采用单线串行传输，就需要约定传输的速率，以按时间区分字符的各位；需要约定串行传输格式，以区分各字符。典型的串行传输格式如图 6-1 所示。

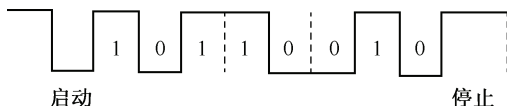


图 6-1 串行传输格式

收发双方需要采用相同的数据传输速率，单位为 bit/s 或 b/s。一旦确定了数据传输速率，也就确定了各代码位之间的时间间隔，即每发送一位代码，其电平（高或低）所维持的时间（即位单元时间）。见图 6-1，高电平为 1，低电平为 0，所发送的字符代码为 10110010。

字符之间用高电平隔开。一个字符的起始标志是一个启动信号。启动信号称为启动位，维持一个单位的低电平。启动位之后是 8 个位单元，可传输 8 位代码。字符的结束标志称为停止位，维持 1.5 单位的高电平。

6.2 键盘及接口

在计算机系统中，键盘是最基本的人机对话输入设备。除了使用标准通用键盘外，工作中也常需自行设计并构成各种专用的小键盘。因此，本节的重点在于掌握形成按键编码的基本原理和方法。

计算机系统本身使用的往往是按标准字键排列的通用键盘。这种键盘包含字符键和一些控制功能键。字符键一般是 ASCII 字符集中的最常用键，如英文字母和数字键，它们的排列符合通常的打字机排列习惯。不同机型的控制功能键的设置可能不同，排列也有所不同。对于汉字键盘，如果采用拼音输入，可直接使用通用的西文键盘；如果采用以字形为基础的输入码，则需对字符键重新定义，定义为以部、首等字形命名的键码。为使有限的键产生更多的按键编码，在键盘上常设有一个换挡键，使一键二用，具有两种键名及相应的键码。

一些专用系统或设备中常需设置专用的键盘。键可能安装在一个小键盘上，或者安装在控制面板上。一般来讲，这类专用键盘的键的数目比较少，因而常被称为小键盘，但键名往往针对具体应用领域的需要来命名。

也可以在 CRT 显示屏幕上示意地显示出键名或者画出键，用户可以通过移动光标寻找并“按键”，还可在屏幕上安装触摸屏，用手或专用的笔在触摸屏上“按键”。常将这种形式称为软键盘。

如何由按键动作产生相应的按键编码呢？一种可能的方法是将按键产生的电信号输入到编码电路，编码器将产生对应的按键编码（在“数字逻辑”课程中已有介绍）。早期曾将这种键盘称为“编码键盘”。当键的数量较多时，编码逻辑的成本较高。直接编码产生键码的方法也不够灵活，一旦编码逻辑电路固定，如果需要重新定义键名和键码，就不够方便。现在，仅当按键数量较少时，或者在形成 8421 码这类很有规律的简单编码时，才采用直接编码逻辑的方法。

在通用键盘上，键的数量较多时，普遍采用扫描方式产生键码。将键连接成矩阵，每个键位于某行、某列交点上，先通过扫描方法找到按下的键的行、列位置，称为位置码或扫描码；再查表（ROM 构成或软件实现）将位置码转换为按键编码。键盘逻辑固定后，某一位置上的键具有固定的位置码；更换转换表的内容，即可重新定义键名和键码。有的参考书将这种键盘称为“非编码键盘”，但我们认为这种提法容易误解为没有对应的键码，因此主张不用这个名称，而称之为“扫描式键盘”。

6.2.1 键盘的类型

键盘的按键可以看成是一种普通的开关，按键或抬键使开关产生不同的信号。按键开关的种类有很多种，产生信号的原理也有较大的差别。

从有无接触点的角度，键盘可分为有触点式、无触点式和虚拟式三大类。有触点式键一般是通过金属（导体）触头把两个接触点接通或断开，以控制信号输入。无触点式是利用磁场变化的霍尔效应或者电流电压引起的电容变化来产生输入信号。虚拟式键则是通过图像投影的方式产生虚拟键盘，再通过光电转换装置或位置感应产生按键信号。

从按键编码的角度，有全编码键盘和非编码键盘两种。全编码键盘是由硬件完成键盘识别功能的，通过识别键是否按下及所按下键的位置，由全编码电路产生一个唯一对应的编码信息（如 ASCII 码）。非编码键盘是由软件完成键盘识别功能的，利用简单的硬件和一套专用键盘编码程序来识别按键的位置，然后由 CPU 将位置码通过查表程序转换成相应的编码信息。非编码键盘的反应速度较慢，但结构简单，并且通过软件能为某些键的重定义提供很大的方便。

按工作原理分，键盘分为机械键盘、塑料薄膜键盘、导电橡胶式键盘和静电电容键盘。

机械（Mechanical）键盘采用类似金属接触式开关，工作原理是使触点导通或断开，具有工艺简单、噪音大、易维护、打字时节奏感强、长期使用手感不会改变等特点。

塑料薄膜式（Membrane）键盘内部共分四层，实现了无机械磨损。其特点是低价格、低噪音和低成本，但是长期使用后由于材质问题手感会发生变化。

导电橡胶式（Conductive Rubber）键盘触点的结构是通过导电橡胶相连，键盘内部有一层凸起带电的导电橡胶，每个按键都对应一个凸起，按下时把下面的触点接通。这种键盘是市场由机械键盘向薄膜键盘的过渡产品。

静电电容（Electrostatic capacitance）键盘利用类似电容式开关的原理，通过按键时改变电极间的距离引起电容容量改变，从而驱动编码器，其特点是无磨损且密封性较好。

1. 接触式

接触式按键有两个触点，按键时闭合，抬键后分离。其具体结构分为以下三种形式。

（1）机械触点式键

如图 6-2 所示，一个机械触点式键由一对触点、键杆、键块、键帽、恢复弹簧等几部分构成。在自然状态时，一对触点处于断开状态，称为常开触点。当键帽被按下时，键块将左边的触点压向右边，使一对触点闭合。手离开键帽后，恢复弹簧使键块上升，恢复自然状态，触点断开。

这种键的结构简单，成本较低，是一种广泛使用的键。其缺点是触点较易氧化，使用寿命相对较短，触点的接触电阻较大，一般用于要求不很高的场合。

（2）薄膜式键

薄膜式键是目前广泛使用的一种短行程触摸式开关，其结构如图 6-3 所示。薄膜是一层很薄且韧性很好的聚酯薄膜，它的背面涂有导电的金属层。基底的正面是印制导电板或者涂有一层导电金属层的柔韧薄膜。基底与薄膜之间用绝缘衬垫隔开，在自然状态下是断开的。按键对应的位置有一个碳心接触点，通过键帽将薄膜按下时，碳心接触点使两个导电层接触。这种键的行程很短，所以也称为触摸式。薄膜式键的结构简单，按键噪音较低，生产成本低，接触层寿命较长，可增加防水处理，可用于恶劣环境；其缺点是使用时间长后，薄膜将逐渐失去弹性。

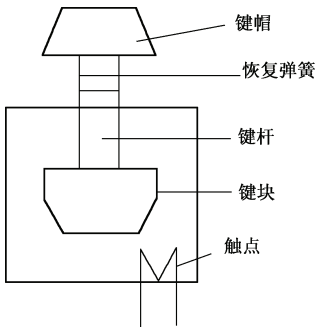


图 6-2 机械触点式键

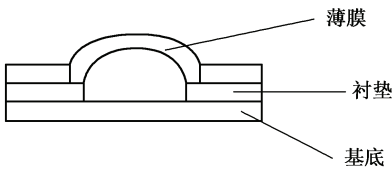


图 6-3 薄膜式键

可将薄膜结构和电容式键的优点结合起来，构成电容薄膜式键。

（3）干簧键

触点焊装在一对簧片上，簧片安置在一个密封的玻璃管内。按键使一个磁铁向下运动，磁力使簧片运动，触点闭合。抬键后，恢复弹簧使磁铁回到原来位置，簧片触点分离。干簧键的触点不易氧化，使用寿命较普通触点式键长，接触也较可靠，但成本较高，一般用于要求较高的专用

场合（键盘上很少使用）。

2. 无触点式

为了提高键的寿命，可应用非接触（无触点）式键。如前所述的分类，常规按键动作方式的无触点式键常见的有以下两种。

（1）电容式键

如图 6-4 所示，电容式键有一个活动极，两个固定极。在自然状态下，活动极与固定极之间的距离较大，极间电容量很小，两个固定极之间为开路状态。当按下键帽，活动极位置下移，与固定极之间仍保持一个很小的间隙（约 0.3 mm），极间电容增大为约 30 pF。此时，从一个固定极（驱动极）输入脉冲信息，可通过极间耦合电容，从另一固定极（检测极）输出，相当于键开关闭合（但并未接触）。换句话说，形成开关动作的是电容量的变化。

电容式键采用类似电容式开关的原理，通过按键改变电极间的距离而产生电容量的变化，暂时形成震荡脉冲允许通过的条件，具有噪音小、磨损小、手感好、工艺复杂的特点。电容式键广泛用于计算机键盘之中。

（2）霍尔效应键

霍尔键是利用霍尔效应产生电信号。按键将驱动一个磁铁下移，造成磁场变化，使霍尔元件产生电位差。在自然状态下，磁铁远离霍尔元件，磁场很弱，没有电信号产生。霍尔键寿命长，可靠，但成本较高，一般用在要求较高的特殊场合。

3. 虚拟式

虚拟式键盘并不是物理存在的，但可以起到常规键盘的输入功效。常见的虚拟式键盘有虚拟激光键盘和触摸屏等。

（1）虚拟激光键盘（Virtual Laser Keyboard, VLK）

虚拟激光键盘最初由 HRR 公司生产，它是一种大小与小型移动电话相仿的虚拟键盘，让用户能像操作普通键盘一样轻易地打出文章或电子邮件。虚拟激光键盘采用光投照技术，几乎能在任意平面上投影出全尺寸的键盘。用在 PDA 和智能手机上时，可方便地进行电子邮件收发、文字处理及电子表格制作。虚拟激光键盘的适用性技术对用户手指运动加以研究，对键盘击打动作进行解码和记录。由于虚拟键盘是光投照所形成的影像，不使用时会完全消失。

图 6-5 展示了一种典型的虚拟激光键盘。

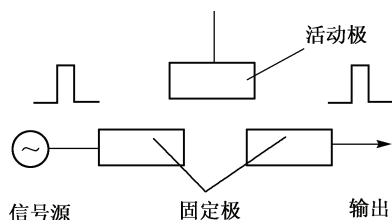


图 6-4 电容式键的原理结构



图 6-5 虚拟激光式键盘

虚拟激光键盘价格十分昂贵，最初 Apple 公司计划在 iPhone 5 上配置这种键盘，但考虑到价格因素和用户的接受度，最终将该计划取消。

美国的谷歌（Google）公司已于 2013 年 1 月向美国专利商标局提交新专利申请，该专利指向激光投射键盘，甚至可以将用户身体变成触控屏。

(2) 触摸屏

随着多媒体技术的发展,触摸屏的应用日益广泛。触摸屏为透明的薄片,安装于显示屏上,与 CRT 或 LCD 等显示器及计算机系统配套使用。显示器显示可供选择的项目,称为键名。当手触摸相应位置时,产生信号,将触摸点的位置坐标送入计算机,可以判别选取的键名。产生信号的原理有多种,常用的有电容式触摸屏(触摸点的电容发生变化)、电阻式触摸屏(压力产生电阻变化)和红外线感应式触摸屏(分辨率较低)。前两种触摸屏的位置分辨率较高,已达到 1280×1024 。严格地说,触摸屏已不是传统的键盘,但能提供另一种形式的按键操作输入方式,我们暂将它归并在有关键操作的一节,简单介绍。

6.2.2 硬件扫描键盘

在键盘上,各键的安装位置可根据操作的需要而定;但在电气连接上,可将所有键连接成矩阵,即分成 n 行、 m 列,每个键连接于某个行线与某个列线之间。通过硬件扫描或软件扫描,识别所按下的键的行列位置,称为位置码或扫描码。如果由硬件逻辑实现扫描,这种键盘称为硬件扫描键盘,或称为电子扫描式编码键盘。所用的硬件逻辑可称为广义上的编码器。

图 6-6 所示硬件扫描式键盘的逻辑组成如下:键盘矩阵、振荡器、计数器、行译码器、列译码器、符合比较器、ROM、接口、去抖动电路等。

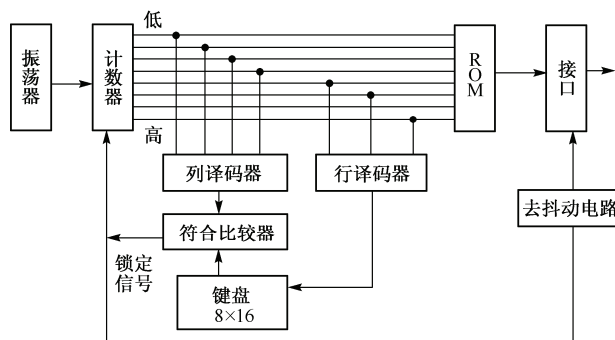


图 6-6 硬件扫描式键盘的逻辑组成

假定键盘矩阵为 8 行 \times 16 列,可安装 128 个键,则位置码需要 7 位,相应地设置一个 7 位计数器。振荡器提供计数脉冲,计数器以 128 为模循环计数。计数器输出 7 位代码,其中高 3 位经译码输出,送键盘矩阵行线。计数器输出的低 4 位送列译码器,列译码器输出送符合比较器。键盘矩阵的列线输出也送符合比较器,二者进行符合比较。

假定按下的键位于第 1 行、第 1 列(序号从 0 开始),则当计数值为 0010001 时,行线 1 被行译码器输出置为低电平。由于该键闭合,使第 1 行与第 1 列接通,则列线 1 也为低电平。低 4 位代码 0001 译码输出与列线输出相同,符合比较器输出一个锁定信号,使计数器停止计数,其输出代码维持为 0010001,就是按键的行列位置码,或称为扫描码。

用一个只读存储器 ROM 芯片装入代码转换表,按键的位置码送往 ROM 作为地址输入,从 ROM 中读出对应的按键字符编码或功能编码。更换 ROM 中写入的内容,即可重新定义各键的编码与功能含义。

由 ROM 输出的键码,经接口芯片送往 CPU。键在闭合过程中往往存在一些难以避免的机械性抖动,使输出信号也产生抖动。抖动发生在前沿部分,对于接触式键,这种抖动可达到数十毫秒。若不避开抖动区,有可能误认为多次按键。因此,在硬件扫描键盘中设置硬件延时电路(如

单稳电路), 延迟数十毫秒之后才识别读取键码, 此时键已稳定闭合。这种电路称为去抖电路。

还需注意一个问题, 即重键问题。当快速按键时, 有可能发生这样一种情况: 前一次按键的键码尚未送出, 后面按键产生了新键码, 造成键码的重叠混乱。图 6-5 所示的逻辑是依靠锁定信号来防止重键现象的。在扫描找到第一次按键位置时, 符合比较器输出锁定信号, 使计数器停止计数, 只认可第一次按键产生的键码。仅当键码送出之后, 才解除对计数器的封锁, 允许扫描识别后面按下的键。当然, 这种暂停扫描的方法只能防止两键重叠, 如果由于 CPU 延缓接收而发生多键重叠, 中间的按键编码就会丢失。所以, 在功能更强的键盘中采取存储多个键码的方法来解决重键问题。

硬件扫描键盘的优点是不需要主机担负扫描任务。当键盘产生键码之后, 才向主机发出中断请求, CPU 以响应中断方式接收随机按键产生的键码。用户已很少用小规模集成电路来构成这种硬件扫描键盘, 而是尽可能利用全集成化的键盘接口芯片, 如 Intel 8279。

6.2.3 软件扫描键盘

可以通过执行键盘扫描程序对键盘矩阵进行扫描, 以识别按键的行列位置, 这种键盘称为软件扫描键盘。

首先要考虑由谁来执行键盘扫描程序。如果对主机工作速度要求不高, 如教学实验用的单板计算机, 可由 CPU 执行键盘扫描程序。按键时, 键盘向主机提出中断请求, CPU 响应后转去执行键盘中断处理程序, 其中包含键盘扫描程序、键码转换程序、预处理程序等。如果对主机工作速度要求较高, 希望尽量少占用 CPU 的处理时间, 可在键盘中设置一个单片机, 由它负责执行键盘扫描程序、预处理程序, 再向 CPU 申请中断送出扫描码。现代计算机的通用键盘大多采用第二种方案。其次考虑如何进行软件扫描。

下面以单板机用的简易键盘和 IBM PC 键盘为例, 介绍两种常用的扫描方法。

1. 逐行扫描法

图 6-7 是一种在单板机中广泛采用的键盘矩阵示意图。16 个字键连接成 4 行、4 列, 4 条列线分别通过上拉电阻接+5 V 电源, 若没有行线的影响, 则列线输出高电平。在执行键盘扫描程序时, CPU 数据输出送往行线, 并将列线输出取回, 判别按键位置。

当有键按下时, 键盘产生中断请求信号, CPU 响应后执行键盘扫描子程序。在单板机监控程序中一般含有扫描子程序, 其流程如图 6-8 所示。

CPU 通过数据线输出代码, 送往行线。从第 0 行开始, 逐行为 0, 其余各行为 1。将列线输出取回至 CPU, 判别其中是否有一位为 0, 是哪一位为 0。假定按下的键将第 1 行第 1 列接通, 则当第 1 行行线为 0 时, 第 1 列列线也为 0, 其余各列线为 1。由此可知按键位置, 即位置码(扫描码), 再查表转换为对应的 ASCII 码。在程序中可插入延时程序, 以避免闭合初期的抖动阶段。上述程序也可由专门的单片机负责执行。

2. 行列扫描法

现在以常用的 IBM PC 键盘为例, 说明一种通用键盘结构、键盘接口及行列扫描法原理。

IBM PC 机的通用键盘采用电容式无触点式键, 共 83~110 键, 连接为 16 行、8 列。采用单片机 Intel 8048 进行控制, 以行列扫描法获得按键扫描码。键盘通过电缆与主机板上的键接口相连。以串行方式将扫描码送往接口, 由移位寄存器组装成并行扫描码, 然后向 CPU 请求中断。CPU 以并行方式从接口中读取按键扫描码。如图 6-9 所示, 虚线左边是键盘逻辑, 右边是接口逻辑。

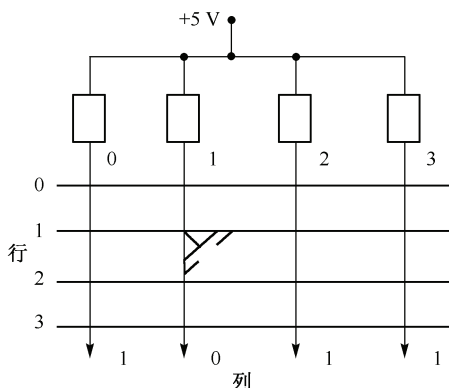


图 6-7 简易扫描式键盘矩阵

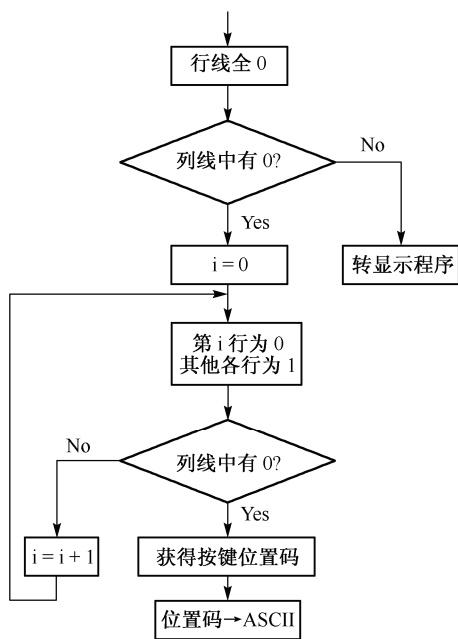


图 6-8 逐行扫描法程序流程

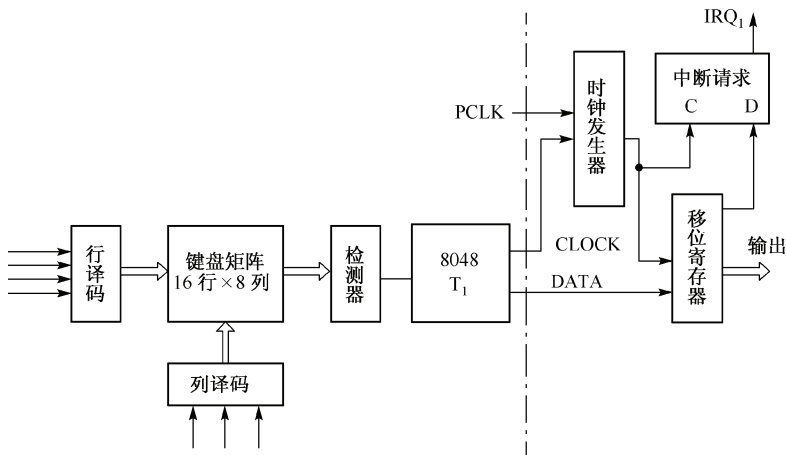


图 6-9 IBM PC 键盘粗框图与接口

键盘接口电路一般在微机主板上，通过电缆与键盘连接，其主要功能如下：① 串行接收键盘送来的按键扫描码；② 将串行扫描码转换为并行扫描码并暂存；③ 向主机发中断请求，通知 CPU 读取扫描码；④ 接收主机发来的命令，传输给键盘，并等候键盘的响应，自检时用来判断键盘的正确性。

单片机 8048 的数据输出分送行译码器与列译码器，可按行列扫描法对键盘矩阵扫描。矩阵输出经检测器，可判别一组行线或一组列线中是否有 1，检测器输出送 8048 的检测端 T_1 。

行列扫描法的工作原理是：先逐列为“1”地步进扫描，由 8048 的 T_1 端测试，列为“1”的行线输出为 1，从而判明按键的列号；再逐行为“1”地步进扫描，判明按键的行号。IBM PC 键盘的操作过程大致如下。

(1) 初始化

封锁 8048 送往接口的时钟信号，禁止键盘工作。并清除键盘接口中的移位寄存器（8 位）和

中断请求触发器，准备接收 8048 送往接口的扫描码。

(2) 允许键盘工作

解除对 8048 送往接口的时钟 CLOCK 的封锁，由系统时钟 PCLK 触发，在时钟发生器输出端产生移位寄存器和中断请求触发器所需的时钟信号。

(3) 扫描键盘

由 8048 输出计数信号控制行、列译码器，先逐列为“1”地步进扫描。当某列为“1”时，若该列线上无键按下，则行线组输出为“0”；若该列线上有键按下，则行线组输出为“1”。将每次扫描结果串行送入 8048 的 T₁ 端，检测某列为“1”时，键盘矩阵行线组输出也为“1”，即表明该列有键按下。再逐行为“1”地步进扫描，由 8048 的 T₁ 端判断某行为“1”时，列线组输出也为“1”，即判断哪行按了键。8048 根据行、列扫描结果便能确定按键位置，并由按键的行号和列号形成对应的扫描码（位置码）。例如，位于键盘第 2 行第 3 列（序号从 0 开始）的键被按下，该位置的序号（从 1 开始）为 50（十进制），对应一个 8 位的扫描码 32H，其中高 4 位表明列线序号（第 3 列），低 4 位表明行线序号（第 2 行）。键盘上每个键位都对应一个扫描码，各键位置对应的扫描码列于表 6-1（第 0 列 0 行交点处不设键）中。

表 6-1 键位与对应的扫描码

| 字键位置 | 扫描码 | 字键位置 | 扫描码 | 字键位置 | 扫描码 | 字键位置 | 扫描码 | 字键位置 | 扫描码 | 字键位置 | 扫描码 |
|------|-----|------|-----|------|-----|------|-----|------|-----|------|-----|
| | | 16 | 10 | 32 | 20 | 48 | 30 | 64 | 40 | 80 | 50 |
| 1 | 01 | 17 | 11 | 33 | 21 | 49 | 31 | 65 | 41 | 81 | 51 |
| 2 | 02 | 18 | 12 | 34 | 22 | 50 | 32 | 66 | 42 | 82 | 52 |
| 3 | 03 | 19 | 13 | 35 | 23 | 51 | 33 | 67 | 43 | 83 | 53 |
| 4 | 04 | 20 | 14 | 36 | 24 | 52 | 34 | 68 | 44 | | |
| 5 | 05 | 21 | 15 | 37 | 25 | 53 | 35 | 69 | 45 | | |
| 6 | 06 | 22 | 16 | 38 | 26 | 54 | 36 | 70 | 46 | | |
| 7 | 07 | 23 | 17 | 39 | 27 | 55 | 37 | 71 | 47 | | |
| 8 | 08 | 24 | 18 | 40 | 28 | 56 | 38 | 72 | 48 | | |
| 9 | 09 | 25 | 19 | 41 | 29 | 57 | 39 | 73 | 49 | | |
| 10 | 0A | 26 | 1A | 42 | 2A | 58 | 3A | 74 | 4A | | |
| 11 | 0B | 27 | 1B | 43 | 2B | 59 | 3B | 75 | 4B | | |
| 12 | 0C | 28 | 1C | 44 | 2C | 60 | 3C | 76 | 4C | | |
| 13 | 0D | 29 | 1D | 45 | 2D | 61 | 3D | 77 | 4D | | |
| 14 | 0E | 30 | 1E | 46 | 2E | 62 | 3E | 78 | 4E | | |
| 15 | 0F | 31 | 1F | 47 | 2F | 63 | 3F | 79 | 4F | | |

(4) 8048 处理重键

8048 的 RAM 中开辟了一个缓冲区，能暂存 20 个扫描码。当多键滚按时，若干按键的扫描码便被放入缓冲区中。按“先进先出”原则，从缓冲区取出扫描码送往接口，就不会出现因快速按键所造成的后按键的扫描码取代先按键的扫描码的情况。

(5) 传输扫描码

按键动作是机械动作，相对于代码传输，速度就慢了。因此，从减少传输线的角度出发，可以只用一根数据线来串行传输扫描码。

8048 在确定某键按下的位置后，由端口 P₂₂ 向移位寄存器的 D₁ 端串行送出 9 位代码，包括 1

位标志和 8 位扫描码。8 位串行扫描码在移位寄存器中由时钟控制依次右移，组装成并行扫描码。1 位标志则由 Q'H 端送入中断请求触发器的数据端，置“1”触发器，产生键盘中断请求 IRQ_1 。在本次中断未响应前，移位寄存器不再接收新的扫描码。

(6) 中断处理

在 IBM PC 系统中，对主要 I/O 设备提供设备一级控制的系统软件称为 BIOS，即基本输入/输出系统。当 CPU 响应键盘中断请求后，执行由 BIOS 提供的键盘中断子程序 KB-INT。该程序首先从接口取出扫描码送入堆栈保存，随后立即清除键盘，使移位寄存器准备好接收新的扫描码，并使中断请求触发器能再次产生中断请求；同时解除对 8048 输出的封锁，允许 8048 送出下一个扫描码。再对收到的扫描码进行识别，并由中断程序通过查表，将扫描码转换为相应键符的标准（或扩展的）ASCII 码，并将转换后的 ASCII 码送入键盘缓冲区。中断处理完毕后返回主程序。

缓冲区的容量足以存放一个快速操作员每秒钟所按的键符。当系统或用户需要键盘输入时，可直接在主程序中以软中断指令（即 INT 16H）的形式调用 BIOS 的键盘 I/O 程序，从缓冲区中取走所需的字符。

6.3 显示设备及接口

6.3.1 显示器的分类

显示器（Display Monitor）也叫监视器，是计算机系统的一种最重要的输出设备之一，它是一种将抽象数据信息通过特定的设备转化成直观的字符或图像的显示工具。

从广义的范畴来看，生活中随处可见的大屏幕、电视机、BSV 液晶拼接的荧光屏、手机和平板电脑等的显示屏都可算在显示器的范畴，然而更多时候，显示器一般是专门指与计算机的主机相连的专用显示器件。

计算机运行的结果往往要以字符或图形的方式在屏幕上显示出来，操作人员才能够直观理解。如果需要对程序做某些修改，人们可以根据显示情况，通过键盘、光笔、鼠标等，将有关数据和命令送入计算机，以便对程序的运行随时进行人工干预和控制。因此，显示设备是一种较为理想的人机交互工具。

对一般用户来讲，要求显示器能在屏幕上指定的位置显示需要的字符、数字和图形。从更深入的层次来讲，如在计算机辅助设计和辅助制造中要求显示设备能绘制三维图形，能实现图形的裁剪、平移、旋转、放大、缩小、投影等几何变换操作。在管理系统中，能够从大表格中提取部分子栏进行处理，开设窗口显示多道程序运行状况等。总之，显示设备与其他 I/O 设备相比较，不仅工作速度快、无机械噪声、灵活轻便，还具有更强的编辑功能和通信功能，被广泛地应用于 CAD、CAM、计算机模拟、过程控制及各种数据处理系统、计算机网络和通信网络中。

显示器屏幕上的字符、图形不能永久记录下来，一旦关机，屏幕上的信息就消失了，所以显示器又常被称为“软拷贝”装置。

显示设备子系统的硬件组成一般包括显示器件（或称为显示器）、控制器和接口。在微机系统中，控制器和接口往往合为一个整体，称为显示器适配卡。其软件组成包含操作系统中的驱动程序，可由操作系统命令调用，能提供专门图形功能的各种图形软件包等。

按照显示器工作原理的不同，常见的显示设备如下。

(1) CRT（Cathode Ray Tube）显示器

CRT 显示器是一种使用阴极射线管的显示设备。阴极射线管主要由 5 部分组成：电子枪

(Electron Gun), 偏转线圈 (Deflection Coils), 荫罩 (Shadow Mask), 荧光粉层 (Phosphor), 玻璃外壳。CRT 显示器在 20 世纪应用非常广泛, 常用在家用电视机、计算机及各类监视设备上。目前, 这类显示器已很少在使用, 基本上退出了历史舞台。

(2) LCD (Liquid Crystal Display) 显示器

LCD 显示器俗称液晶显示器。LCD 显示器的基本工作原理是: 在显示器内部有很多液晶粒子, 它们有规律地排列成一定的形状, 并且每面的颜色都不同, 分为红色、绿色和蓝色。这三原色能还原成任意颜色, 当显示器收到计算机需显示的信息时, 控制每个液晶粒子转动到不同颜色的面, 组合形成不同的颜色和图像。

LCD 显示器的优点是辐射弱、体积小, 缺点是色彩不如 CRT 显示器丰富, 可视角度也不如 CRT 显示器广等。目前, LCD 显示器已经在很多领域中全面取代了 CRT 显示器, 成为了业界最主流的显示器之一。

(3) LED (Light Emitting Diode) 显示屏

LED 即发光二极管。LED 显示屏是一种通过控制半导体发光二极管来显示信息的设备, 可以显示文字、图形、图像、动画、行情、视频、录像信号。

早期, LED 只是用于微型指示灯, 在计算机、音响和录像机等高档设备中应用。随着大规模集成电路和计算机技术的不断进步, LED 显示器正在迅速崛起, 逐渐扩展到证券行情股票机、数码相机、PDA 及手机领域。LED 显示器集微电子技术、计算机技术、信息处理于一体, 以其色彩鲜艳、动态范围广、亮度高、寿命长、工作稳定可靠等优点, 成为最具优势的新一代显示器件, 在大型广场、商业广告、体育场馆、信息传播、新闻发布、证券交易等领域应用得十分广泛。

(4) PDP (Plasma Display Panel) 显示器

PDP 即等离子显示器, 是一种采用等离子平面屏幕技术的新一代显示设备。PDP 显示器的成像原理是, 在显示屏上排列着数量众多的微小低压气体室, 当通过电流时, 能激发并使这些气体室发出肉眼看不见的紫外光, 然后控制紫外光撞击后面玻璃上的红、绿、蓝三色荧光体, 使其发出可见光, 据此成像。

PDP 显示器的优点是: 厚度薄、亮度高、分辨率高、体积小等, 常作为家用的壁挂式电视机使用, 可能代表了未来计算机显示器的发展趋势。

(5) 3D 显示器

与一般显示器不同, 3D 显示器能显示出立体感很强的画面, 业界一直认为它是未来显示技术发展的终极目标。从 20 世纪开始, 有很多企业和研究机构从事 3D 显示器的研究。日本、欧美、韩国等发达国家早在 80 年代就涉足 3D 显示技术的研发, 并于 90 年代开始陆续取得不同程度的研究成果, 现在已逐步发展出需佩戴立体眼镜和无需佩戴立体眼镜的两个 3D 显示技术分支体系。

传统的 3D 电影在荧幕上有两组图像 (来源于在拍摄时的互成角度的两台摄影机), 观众必须戴上偏光镜才能消除重影 (让一只眼只接收一组图像), 形成视差 (Parallax), 从而让观众产生立体感。另一种是“真 3D”显示, 通过自动立体 (Auto-stereoscopic) 显示技术来实现。这种技术利用“视差栅栏”, 使观众的两只眼睛分别接收不同的图像, 从而形成立体视觉效果。平面 3D 显示器要形成立体感强的影像, 技术上至少要提供两组相位不同的图像才能实现, 其中快门式 3D 技术和不闪式 3D 技术是目前 3D 显示器中最常用的两种技术。

6.3.2 显示器的成像原理

在介绍显示器基本工作原理之前, 先介绍几个基本概念。

1. 图形和图像

图形和图像是现代显示技术中常用的术语。

图形(Graphic)最初是指没有亮暗层次变化的线条图,如建筑、机械所用的工程设计图、电路图。早期的图形显示和处理只是局限在二值化的范围,只能用线条的有无来表示简单的图形;图像(Image)最初是指具有亮暗层次的图,如自然景物、新闻照片等。经计算机处理后显示的图像称作数字图像,就是将图片上连续的亮暗变化转换为离散的数字量,并以颜色光点阵列的形式显示输出。

在显示屏幕上,图形和图像都是由称作像素(Pixel)的光点组成的。现在的图形也可以有颜色、深浅层次的变化。但图形学和数字图像处理是两个不同的学科,它们所研究的问题不同,应用领域不同,使用的技术方法不同,图形和图像的输入手段也不同。

图形学的主要任务是研究如何用计算机表示现实世界的各种事物,并且形象、逼真地进行显示,如动画设计、花布图案设计、地图的显示等平面图,飞机、汽车、建筑物的造型设计等立体图。这些图的显示效果要有真实感,需要有深浅和颜色。图形学所用的技术包括点、线、面、体等平面和立体图的表示与生成。同时,由于要在平面上显示立体图,还要研究阴影的产生,隐藏线、隐藏面的消除技术以及光照方向与颜色的模拟等技术。

数字图像处理所处理的对象多半来自客观世界,如由摄像机摄取下来存入计算机的数字图像(遥感图像、医用图像等)。由于图形可以按人的意志描绘,所以无噪声干扰,而且规则整齐,富有创造性。图像则可能充满噪声,很不清晰。由于摄取的位置随机,图像可能发生畸变。图像处理的任务是去除噪声,恢复原形,使之清晰,并从中抽取有用的信息,以供观察。图像主要用摄像机输入,经数字化以后逐点存储,因此图像需要占用非常庞大的主存储空间。在计算机中表示图形,则只需存储绘图命令和坐标点,没必要存储每个像素点。例如,用两点式表示一条直线为 $L(x_0, y_0; x_1, y_1)$,计算机中只存储了这个表达式,坐标点到图形像素点的转换由计算机或显示设备自动完成。

2. 分辨率和灰度级

分辨率是指显示器单屏所能表示的像素个数,相同面积显示屏幕上的像素点密度越大,则显示器的分辨率也就越高,图像也就越清晰。

显示器的分辨率取决于显像管成像光点的粒度、屏幕尺寸等因素,同时还要求显示缓冲存储器(刷新存储器)要有能与显示像素数相匹配的存储空间,以用来存储每个像素的信息。

例如,12英寸彩色显示器的分辨率为 640×480 像素。每个像素的间距为 0.31 mm ,水平方向的640像素所占显示长度为 198.4 mm ,垂直方向480个像素按4:3宽高比例分配($640 \times 3/4 = 480$)。按这个分辨率表示的图像具有较好的水平线性和垂直线性,否则看起来会失真变形。同样,16英寸的显示器显示 1024×768 个像素也满足4:3宽高比。某些专用方形显示分辨率可达 1024×1024 甚至更高。

灰度级是指黑白显示器中所显示的像素点的亮暗差别,在彩色显示器中则表现为颜色的不同。灰度级越大,图像层次越清楚逼真。灰度级取决于每个像素对应刷新存储器单元的位数和显示器本身的性能。如果用4位表示一个像素,则只有16级灰度或颜色,如果用8位表示一个像素,则有256级灰度或颜色。字符显示器只用“0”、“1”两级灰度就可表示字符的有无,故这种只有两级灰度的显示器称为单色显示器。具有多种灰度级的黑白显示器称为多灰度级黑白显示器。图像显示器的灰度级一般在256级以上。

3. 颜色深度和颜色数

颜色深度一般是指用来表示一个像素的二进制代码长度，如颜色深度为 24 位表示一个像素用 24 bit 来标示，有时也称之为 24 位色。

颜色数是指单个像素可以显示出的不同颜色种类数量，它和颜色深度直接相关。例如，颜色深度为 24 位色，则其对应的颜色数为 $2^{24}=16\text{M}$ ，可理解一个像素可以显示出 16M 种不同的颜色。颜色数越多，则显示出的画面色彩也就越丰富，画质也更艳丽。

4. 刷新频率和显示存储器

显示器上显示的内容只能维持短暂时间，为了使人眼能在显示器上看到持续稳定的图像，必须使屏幕上显示的内容在彻底消失之前再次重新显示，这个过程就叫做屏幕的刷新。单位时间内（通常指 1 秒）屏幕内容刷新的次数，就是屏幕的刷新频率，单位赫兹（Hz）。

按人的视觉生理特性，刷新频率一般大于 25 Hz 时，人眼就不会感到闪烁。早期的 CRT 显示设备中通常选用 75 Hz 的刷新频率（帧频），即每秒刷新 70 帧图像。而现在主流的 LCD 显示器，则采用 60 Hz 的刷新频率，就能保持显示稳定的图像。家用彩色电视机的刷新频率一般为 50 Hz，就能提供稳定的图像画质。

为了持续不断地提供刷新图像所需的数据信号，必须把一帧图像对应的数据信息存储在一个专用的刷新存储器中，这个存储器也叫显示存储器，一般由随机存储器构成，因此简称显存（Video Random Access Memory，VRAM）。

显示器一方面对屏幕进行扫描，另一方面同步地从显存中读取需显示的内容，送往显示器件。因此，对显存的操作是显示器工作的软、硬界面所在。从软件角度讲，执行显示软件的最终结果是向显存写入显示信息。为了在指定的屏幕位置显示某个字符，就需向显存的相应单元写入该字符或像素编码。为了更新屏幕显示内容，就需相应地刷新显存的内容。为了使画面呈现某种动画效果，就需要使显存中的内容作相应变化，或者在读取时进行某种地址转换。

从硬件角度讲，显示器控制器的基本任务就是将显存内容同步地送往显示屏幕。为此，需要决定何时发出水平同步信号？何时发出垂直同步信号？何时访问显存？地址码如何产生？从显存中读出的代码是否要经过一定的转换以变为控制光点的信号？等等。这些工作以一定频率，同步地、周而复始地进行。采取不同显示规格，上述关系将不同。

显存中的内容一般包含显示内容和属性内容两个部分。前者提供显示字符代码或图像的像素信息，后者则提供与之相关的属性信息。这两部分内容可分别存放在两个缓冲存储器中，一个可被称为基本显示缓存，另一个可被称为属性缓存。常将这两个存储体统一编址，一个为偶数地址，另一个为奇数地址。也可以将两部分存放在一个存储器中，依靠地址码为偶数或奇数进行区分。

显存一般设置在显示器的控制适配器中，如 CGA 卡、EGA 卡、VGA 卡。在有些集成显卡的微机中，显存一般占用部分主存空间，从软件上讲视为主存的一部分。在带独立显卡的显示终端中，显存作为外围设备存在，通常与主存分离。

作为显示系统的重要组成部分，显存随着显示芯片的发展也一直在快速发展。早期普遍采用标准的 EDORAM 和 SDRAM 显存（已淘汰），目前已广泛使用 DDR 显存（DDR2 和 DDR3），甚至已开始出现更高端的 GDDR5 显存。

在了解各种显示器的工作原理时，应当紧紧抓住屏幕显示与显存之间的对应关系这一条基本线索，即显示缓存中存放的内容、所需容量大小、缓存地址组织、由字符编码到字符点阵之间的转换关系，以及如何通过一组同步计数器控制屏幕的水平扫描与垂直扫描、何时访问显存，等等。

(1) 显存的内容和容量

显示器通常能以字符/数字和图形/图像这两种显示模式进行工作。

当工作于字符/数字模式时，在缓存 RAM 中存放的是一帧待显示的字符的 ASCII 编码（也可能是其他形式的编码），字符的点阵信息则放在字符发生器（字库）中。在这种方式下，一个字符的编码占显存的 1 字节，此时显存的容量是由屏幕上字符的行列规格来决定的。例如，一帧字符的显示规格为 25 行×80 列，那么 VRAM 的最小容量是 25 行×80 列×1 B=2 KB。

缓存的容量也可以大于一帧字符数，用来存放几帧字符的编码。在这种情况下，通过控制缓存的指针就可以在屏幕上显示不同帧中的字符内容，实现屏幕的“硬件滚动”。

如果工作于图形/图像模式，那么，显存中保存的内容就是一帧待显示的图形的像素信息编码，不同的代码表示图形中的不同的颜色像素。这些图形可以是几何图形、任意曲线图形、汉字或字符。这里需要特别说明的是，在图形方式下字符的点阵是以位图的形式直接存放在显示缓存中的，因此字符可按像素为单位在屏幕的任意位置上显示。这一点与字符方式是不同的，字符显示的缓存中存放字符的 ASCII 码，信息安排比较紧凑整齐，便于进行编辑修改操作，但格式死板。

图形方式中，缓存的容量不仅取决于屏幕分辨率的高低，还与显示的颜色种类有关。在单色显示（黑白显示）时，图形的每个点一般只用一位二进制代码来表示。该点若为白色（亮点），该位代码取 1；若为黑色（暗点），代码取 0。因此，缓存的一个字节可以存放 8 个点。很显然，分辨率越高，缓存的容量就越大。例如，屏幕分辨率为 200 线×640 点，那么显存的最小容量是 200 线×640 点×1 bit/8=16 KB。

在彩色显示或灰度显示时，每个像素点需要若干位代码来表示。例如，显示器分辨率为 1024 线×1024 点，能显示 256 级灰度图，则显存的基本容量应为 1024 线×1024 点×8 bit/8 = 1 MB。

如果显示器的分辨率不变，显存的容量增加，则能显示的颜色数也会增加。反之，若显存的容量不变，提高显示器的分辨率，则能显示的颜色种类将减少。

(2) 属性信息与属性缓存

在某些情况下，人们希望屏幕上的字符能够闪烁，以作提示；或字符下画一横线，表示强调；或背景与字符着上不同的颜色，使显示效果更为生动等。我们称这些特色为字符的显示属性。屏幕上每个字符的属性信息可以存放在 1 字节中，因此需要一个与显示缓存容量相同的存储器来存放一帧所有字符的属性信息，这个存储器称为属性缓存。例如，用来存放字符的显示缓存若为 2 KB，那么相应的属性缓存也应该有 2 KB 的容量。

在早期的低端微机中，将 32 KB 的显示缓存和 32 KB 属性缓存共 64 KB 空间合在一起编址。其中，偶地址字节放字符代码，奇地址字节放属性代码。在黑白显示和彩色显示下，属性字节的内容并不完全相同，表 6-2 给出了两种方式下属性字节的信息。

从以上分析可以看出，分辨率、颜色数与显存容量密切相关。对于字符显示方式，如分辨率为 C 列× L 行，而一个字符的编码与属性、颜色数共占 n 字节，则显存的总容量应不少于 $C \times L \times n$ 字节。对于图形显示方式，如果分辨率为 $C \times L$ 像素，而每个像素的颜色数用 n 位二进制代码表示，则显存容量应不少于 $C \times L \times n$ 位。两种显示方式的 C 、 L 值不同，显然，图形方式所需的显存容量一般都大于字符方式。

如果一台显示器既可用工作于字符方式又可工作于图形/图像方式，且各有多种分辨率规格，则显存的基本容量计算应以图形/图像方式最高分辨率为准。此外，显存的存取周期还必须满足刷新频率的要求，因此存储容量和存取周期也是显存的重要指标。

表 6-2 字符显示的属性字节与属性代码

| (a) 黑白显示下的属性字节与属性代码 | | | | | | (b) 彩色显示下的属性字节与属性代码 | | | | | | | | | | | |
|---------------------|----|--|--------|----------------|----|--|---------|----------------|-----|--|--|---|--|----------|--|-------|--|
| D ₇ | | D ₆ D ₅ D ₄ | | D ₃ | | D ₂ D ₁ D ₀ | | D ₇ | | D ₆ D ₅ D ₄ | | D ₃ D ₂ D ₁ D ₀ | | | | | |
| 奇地址 | 闪烁 | | 底色（背景） | | 增辉 | | 显示色（前景） | | 奇地址 | 闪烁 | | 底色显示色 | | 前景彩色 | | | |
| | | | | | | | | | | | | | | | | | |
| 底色 | | 显示色 | | 显示效果 | | 闪烁 | | 增辉 | | 前景彩色 | | 颜色 | | 背景彩色 | | 闪烁 | |
| 000 | | 000 | | 不显示（黑） | | 0 不闪烁 | | 0 正常 | | 0000 | | 黑 | | 与彩色选择 | | 0 不闪烁 | |
| 000 | | 001 | | 正常显示加下横线 | | 1 闪烁 | | 1 加亮 | | 0001 | | 蓝 | | 器提供的背景 | | 1 闪烁 | |
| 000 | | 111 | | 黑底白字 | | | | | | 0010 | | 绿 | | 色、亮度信号 | | | |
| 111 | | 000 | | 白底黑字 | | | | | | 0111 | | 白 | | 组合，选择 16 | | | |
| 111 | | 111 | | 白色块 | | | | | | 1111 | | 强白 | | 种颜色之一 | | | |

5. 随机扫描和光栅扫描

电子束在荧光屏上按某种轨迹运动称为扫描 (Scan), 控制电子束扫描轨迹的电路称为扫描偏转电路。主要扫描方式有两种, 随机扫描和光栅扫描。

随机扫描是控制电子束在 CRT 屏幕上随机地运动, 从而产生图形和字符。电子束只在需要作图的地方扫描, 而不必扫描全屏幕, 所以这种扫描方式画图速度快, 图像清晰。

光栅扫描是 CRT 显示器中采用的扫描方法。CRT 显示器要求图像充满整个画面, 因此要求电子束扫过整个屏幕。光栅扫描是从上至下顺序扫描, 采用逐行扫描和隔行扫描两种方式。逐行扫描就是从屏幕顶部开始一行接一行地扫描, 一直到底, 反复进行。CRT 显示器系统采用隔行扫描, 它把一帧图像分为奇数行和偶数行, 如果本次扫描奇数行, 下次则扫描偶数行, 交替进行。

由于 CRT 显示器技术历史久、技术成熟, 所以计算机的 CRT 显示器广泛采用光栅扫描方式。其主要优点是显示部分易于配套, 易于维修, 甚至可用家用 CRT 电视机作为“显示头”。但其主要缺点是显示时冗余时间多, 分辨率不如随机扫描方式高, 图形的直线和圆弧不够光滑。尽管如此, 光栅扫描上个世纪仍然是显示设备采用的主流技术。目前, 主流的 LCD 显示器已基本淘汰了这种光栅扫描方式。

通常, 根据显示器工作模式的不同, 一般可以粗略地划分为两种方式, 即字符/数字模式和图形/图像模式, 这两种模式的成像原理存在很大不同。下面分别介绍这两种显示原理。

1. 字符/数字模式的显示原理

在这种显示方式中, 以字符为显示内容的基本单元, 又称为文本显示方式。实际上, 字符是由点阵组成的, 工作在字符/数字模式的显示器对应显存中保存的是字符/数字的 ASCII 代码, 因此显示过程中需要将显存中读取的 ASCII 码转换为字符点阵代码再显示。

显示器工作在字符/数字模式下时, 其显存只取决于屏幕上所需显示的字符/数字的数量。例如, 300×400 的字符/数字显示器, 屏幕上可显示 300 行×400 列=120 000 个字符, 每个字符在显存中对应 1 字节数据, 则基本显存容量应为 120000×1 B≈120 KB。

一台显示器可显示的字符种类与字符点阵规格, 决定了字符发生器 ROM 的容量大小, 而显存的容量则与此无直接关系。

进一步描述字符/数字模式的指标是每个字符/数字在屏幕上显示时, 字符点阵的组成情况, 即每个字符点阵由横向多少个像点 (像素) 数与纵向多少个像点 (像素) 数组成。通常, 还需要考虑字符之间的横向间隔, 以及各行字符之间的纵向间隔。

常用的字符点阵结构有 5×7 点阵, 5×8 点阵, 7×9 点阵等。所谓 5×7 点阵, 是指每个字符由

横向 5 个点、纵向 7 个点共 35 个点组成。在显示到屏幕上时，点阵中需要显示的部分为亮点（颜色），不需要显示的部分则为暗点。字符点阵结构所包含的点数越多，所显示的字符就越清晰，且字符曲线表示得越平滑和越逼真，所以用 7×9 点阵可以用来形象地显示小写字符。

在显示设备中，用来根据从显存中读取的字符/数字 ASCII 编码产生字符点阵图形的器件称为字符发生器。字符发生器一般由专用存储芯片构成，如 Apple-I 所用的 2513 芯片采用 5×8 点阵，能产生 64 种字符的点阵信息。通常，也可以用通用 ROM 作为字符发生器，如在微机上，可用 8 KB ROM 空间来存放 256 个字符的点阵代码，有三套字体，即每个字符可采用 7×9、7×7、5×7 点阵。

图 6-10 是 2513 芯片的逻辑框图，该芯片的核心部分是一个 ROM，能存储 64 种字符的点阵码。每个字符排成 5 列×8 行的点阵形式，如图 6-11(a)所示。在点阵图中，字符（如 H）需要显示的点（亮点）均用代码 1 表示，不需要显示的点（暗点）则用代码 0 表示，如图 6-11(b)所示。每个字符都以这种点阵图形的代码形式存放在 ROM 中，一行代码占 1 个存储单元，因此一个字符的点阵代码占 8 个存储单元，每个单元只需 5 位。

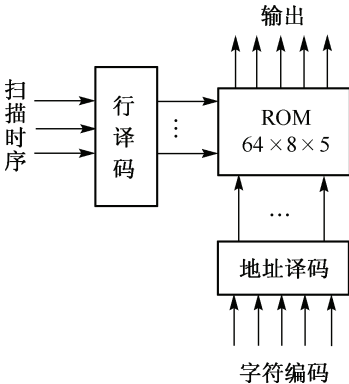


图 6-10 2513 字符发生器的逻辑框图

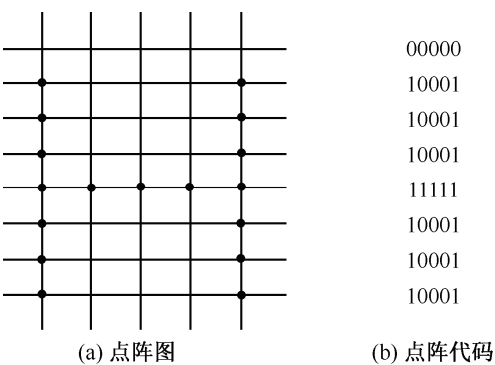


图 6-11 5×8 字符点阵图形与点阵代码

64 个字符以各自编码的 6 位作为字符发生器的高位地址，当需要在屏幕上显示某字符时，按该字符的 6 位编码访问 ROM，选中这一字符的点阵。该字符的点阵信息是按行输出的，并且要与扫描保持同步。因此，可用显示控制器提供的扫描时序（扫描线序号）作为字符发生器的低 3 位地址，经译码后依次取出字符点阵的 8 行代码。

例如，字符 H 的 6 位 ASCII 码是 001000，以这 6 位编码作为高位地址访问 ROM，选中字符 H 的点阵代码。当控制器的扫描线序号为 000 时，即扫描屏幕上该字符位置的第一行时，字符发生器输出该点阵图形的第一行代码；序号为 001 时，输出第二行点阵代码，……，直到序号为 111，即扫描字符位置的最后一行时，字符 H 的 8 行代码全部输出完毕。

在屏幕上，每行一般要显示多个字符。在进行全屏显示时，并不是对一排的每个字符单独进行点阵输出（即输出完一个字符的各行点阵，再输出同排下一字符的各行点阵），而是采用对一排的所有字符的点阵进行逐行依次输出的方式。例如，欲显示某字符行字符 ABC……T，当扫描线序号为 000 时，显示电路根据各字符编码依次从字符发生器中取出 A、B、C、……、T 各字符的第一行点阵代码输出，并在屏幕上的当前行第一条扫描线位置上显示出这些字符的第一行点阵；再扫描本行字符的下一行点阵代码，依次取出该排各字符的第二行代码，并在屏幕上显示出它们的第二行点阵。如此循环，直到显示完该行字符的全部点阵，那么每个字符的所有点阵（如 8 行点阵）便全部显示在相应的位置上，屏幕上也就出现了一排完整的字符。当显示下一排字符时，重复上述字符点阵输出过程。

如前面所述，为了使屏幕上显示的字符不挤在一起、易于辨认，每行中各字符之间要留出若干点的位置，作为字符间的横向间隔，这些点不能显示（消隐）；在各行字符之间也要留出若干条扫描线位置作为字符行间的间隔，这些位置上点也是不能显示（消隐）的。例如，微机的显示器

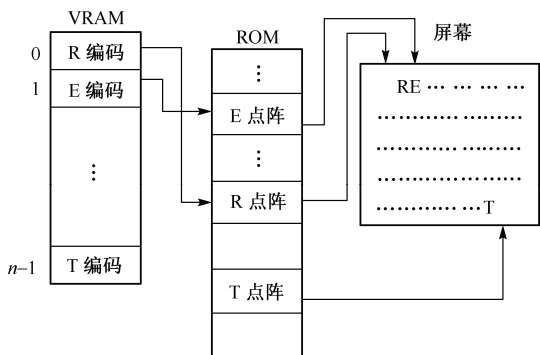


图 6-12 字符/数字显示模式

器一般采用 7×9 有效字符点阵，字符所占区间为 9×14 点阵。换句话说，字符之间的横向间隔是 2 个消隐点宽度，各行字符之间的间隔是 5 条消隐线高度。

注意，工作在字符/数字模式的显示器的屏幕显示的基本控制单位仍然是像素（像点），此时的显示原理可以归纳为如图 6-12 所示。

屏幕上每一个字符位置对应显存中 1 字节的字符编码，各字节单元的地址按屏幕由左向右、自上而下的显示顺序从低向高安排。也就是说，缓存 0 号单元所放的字符数据经字符发生器转换为字形点阵后，显示在屏幕第一排字符左边第一个位置上，1 号单元放的字符数据转换后显示在屏幕第一行左边第二个位置上……缓存最后一个单元放的字符数据转换后显示在屏幕最后一排右边最末一个位置上。

字符/数字模式的显示成像原理可以笼统地概括为：从 VRAM（显存）中读取字符编码（ASCII 码），根据字符编码定位 ROM（点阵发生器），再根据点阵的行控制信号，控制各行字符相同行号的点阵代码输出，再用输出的各行点阵代码形成视频控制信号，去控制屏幕像点（像素）的明暗及颜色变化，从而将字符最终显示在屏幕上。

2. 图形/图像模式的显示原理

根据数字图像的基本原理，一个像素显示出的视觉效果，主要取决于三基色 R（Red，红色分量）、G（Green，绿色分量）、B（Blue，蓝色分量）和其他参数。例如，以 24 位真彩色图像为例，RGB 为(0, 0, 0)，对应像素便显示为黑色；RGB 为(255, 0, 0)，像素变成红色；若 RGB 为(0, 255, 0)，像素显示为绿色；RGB 为(255, 255, 255)，此时像素又显示成为了白色。

在 24 位的真彩图像中，每个分量均由 8 bit 数据来表示，因此 R、G、B 这三个分量一共能组合成 224 种颜色，其中一部分对应关系如表 6-3 所示。

表 6-3 部分 R、G、B 分量与颜色的对应关系

| R（红） | G（绿） | B（蓝） | 组合颜色 | R（红） | G（绿） | B（蓝） | 组合颜色 |
|------|------|------|------|------|------|------|------|
| 0 | 0 | 0 | 黑色 | 255 | 165 | 0 | 橙红色 |
| 0 | 0 | 255 | 蓝色 | 0 | 0 | 139 | 深蓝色 |
| 0 | 255 | 0 | 绿色 | 144 | 238 | 144 | 淡绿色 |
| 0 | 255 | 255 | 青色 | 0 | 139 | 139 | 深青色 |
| 255 | 0 | 0 | 红色 | 0 | 100 | 0 | 深绿色 |
| 255 | 0 | 255 | 紫红 | 255 | 140 | 0 | 深橙色 |
| 255 | 215 | 0 | 金色 | 173 | 255 | 47 | 绿黄色 |
| 255 | 255 | 255 | 白色 | 259 | 192 | 103 | 粉红色 |

显示器工作在图形/图像模式时，屏幕显示主要就是利用三基色控制像素点，使像素显示为不同的颜色。屏幕上显示为不同颜色的像素，一起就构成了显示的一帧画面。

如图 6-13 所示，在图形/图像模式时，与字符/数字模式不同，此时 VRAM（显存）中保存的是屏幕像素对应的二进制 RGB 颜色编码。此时，直接从 VRAM 中读取当前像素位置所在 VRAM 中对应的 RGB 编码，并将 RGB 编码经过一系列转换，形成视频信号，视频信号再用来直接控制对应像素点的 RGB 分量值，从而使像素显示为一定的颜色。全部像素显示完成后，屏幕上的一帧画面也就显示完整了。这种模式下，显存地址组织一般也按屏幕的像素位置由低向高递增，低地址存放的点先显示，高地址存放的点后显示。

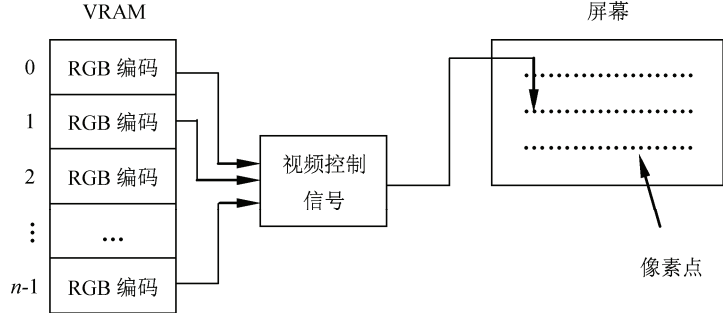


图 6-13 图形/图像显示模式

6.3.3 CRT 显示器

CRT 显示器学名为“阴极射线显像管”，是一种使用阴极射线管（Cathode Ray Tube）的显示器，它的结构原理如图 6-14 所示，显示器的外观如图 6-15 所示。

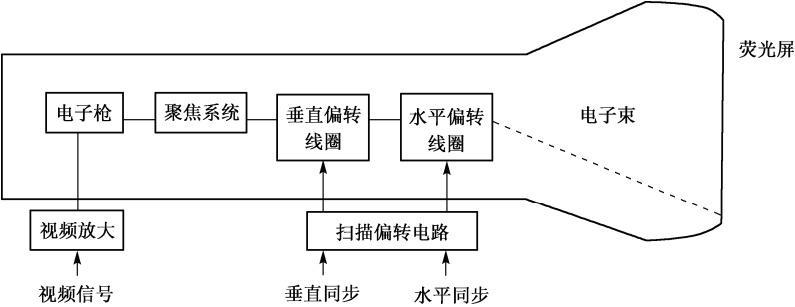


图 6-14 显示头结构原理



图 6-15 CRT 显示器的外观

CRT 显示器主要由电子枪、视频放大系统、扫描偏转系统、荧光屏等几部分组成。人们之所以能在 CRT 荧光屏上看见所显示的字符或图形，是因为阴极射线管的电子枪所发射的电子流经聚焦后形成电子束，轰击荧光屏，使屏上所涂的荧光粉发出可见光。要在屏幕上指定的位置进行显示，需要通过扫描偏转系统产生两个互相垂直的电磁场，控制电子束在 X 方向和 Y 方向偏转，从而将电子束引向屏幕的相应位置。

CRT 显示器可以采用多种扫描方式进行工作，但用得最多的是光栅扫描和随机扫描两种方式。在随机扫描方式中，电子束没有固定的扫描路径，只在要显示字符或图形的地方扫描，因此扫描控制信号随显示内容的不同而有所变化，这就使扫描电路比较复杂。光栅扫描有固定的格式，不管屏幕上需要显示或不需显示的地方，都按统一路径全屏幕扫描，因此扫描控制信号不随显示画面变化，使得扫描电路比较简单。目前，随机扫描方式只用于图形显示，光栅扫描因控制简单，被广泛用于字符显示和图形显示器中。

1. 光栅的扫描

在光栅扫描方式中,电子束从荧光屏的左上角开始,沿着稍稍倾斜的水平方向匀速地向右扫描,到达屏幕右端后迅速水平回扫到左端下一行位置,又从左向右匀速地扫描。这样一行一行地

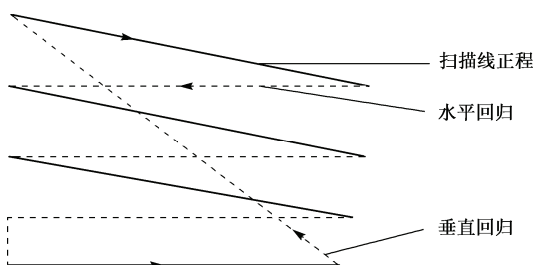


图 6-16 光栅-电子束的运动轨迹

扫描,直到屏幕最后一行的右端。然后垂直回扫,返回屏幕左上角,重复前面的扫描过程。经过电子束如此反复地从左至右、自上而下的全屏扫描,便在荧光屏上形成了一条一条的垂直分布于整个屏幕的水平扫描线,如图 6-16 所示。这些扫描线称为光栅,代表了电子束在屏幕上的运动轨迹。在进行水平回扫和垂直回扫时,荧光屏上不出现亮线,CRT 此时处于“消隐”(Blank)状态,如图 6-16 中的虚线所示。

为了实现这种有规律的光栅扫描,应该通过电磁场的变化,控制电子束既做水平方向的运动,也做垂直方向的运动。因此,需要在 CRT 的水平偏转线圈和垂直偏转线圈中,分别通以按不同频率作线性变化的锯齿波电流或加锯齿波电压,如图 6-17 所示。水平方向的锯齿波扫描电流引起的电磁场变化控制电子束的运动,形成水平扫描线(行扫描);垂直方向的锯齿波扫描电流引起的电磁场变化则造成扫描线的垂直移动(场扫描)。这些锯齿波电流是由 CRT 控制器送来的水平同步和垂直同步信号触发扫描电路中的锯齿波发生器产生的。

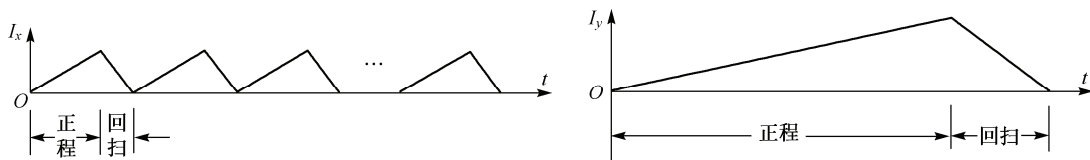


图 6-17 锯齿波扫描电流

如图 6-17 所示,在场扫描的一个垂直周期内,行扫描扫完一帧,行、场扫描频率之比正好是一帧所完成的扫描线的条数。例如,若每帧光栅扫描 625 线,则有关系式 $f_x = 625f_y$ 。其中, f_x 为行扫描频率, f_y 为场扫描频率。

光的亮度要随时间衰减,为了在屏幕上得到稳定的不闪烁的图像,要求一帧画面每秒钟内必须反复显示若干次,一般为 25 遍以上,因此帧频应该不低于 25 Hz。

光栅扫描可以采用逐行扫描或隔行扫描的方法。在逐行扫描中,一帧图像的 625 条光栅只需一遍就能扫描完。我们把扫描一遍称为一场,因此逐行扫描也称为“一帧一场”。显然,在这种方法中,场频和帧频是相等的。若采用隔行扫描,需要将一幅图像的扫描分两遍完成,称为“一帧两场”。第一遍称为奇数场,扫描一帧图像的所有奇数行光栅;第二遍称为偶数场,扫描一帧图像的所有偶数行光栅。因此,每场扫描的光栅行数只有一帧一场的二分之一,即 312.5 行。若按帧频 25 Hz 计算,则在一帧两场的方法中每秒钟要扫描 50 场,因此场频为帧频的 2 倍。

早期流行的 CRT 图形显示器一般采用逐行扫描方法,其场频(帧频)一般为 85 Hz,才能使显示的画面更稳定。

2. 一帧画面的组成

由前述可知,一帧画面是由一定数量的平行的水平扫描线组成的,这些扫描线在一个垂直的场扫描控制下均匀地、自上而下地分布于整个画面。

每条扫描线由若干像素组成。每个像素的位置和亮度取决于下面的基本因素。电子束 X 方向和 Y 方向的偏转决定像素的位置，电子束的通、断、强、弱则决定像素的亮度。我们已经知道，由水平同步和垂直同步信号经扫描电路产生的行、场扫描电流能够通过电磁场的变化控制电子束的 X 偏转和 Y 偏转，那么电子束的强弱由什么信号来控制呢？前面已经讲过，电子束由 CRT 管电子枪中的阴极发射的电子流经聚焦而成。在电子枪中还设有一个控制栅，用控制栅和阴极之间的电位差来控制电子束电流的大小。因此，可以将控制像素亮度的视频信号（亮度信号）通过视频放大电路加在控制栅上，这样便可以用外部信号来调整控制栅极电位的高低，以控制电子枪所发射的电子束的强弱，从而使像素的亮度发生变化。例如，当控制栅所加的信号为“1”时，电子枪发射具有一定能量的电子束，使屏幕上相应的点发亮；当信号为“0”时，电子枪不发射电子束，使像素成为暗点。若改变所加控制信号的电平大小，则可控制像素具有不同的亮度（灰度）。

如果显示的画面是彩色的，那么每个像素还要受各颜色分量的控制。彩色 CRT 显示器根据三基色原理，在显示头中设有三个电子枪，分别发射能产生红、绿、蓝三种基色的电子束。它们受三套视频放大电路的控制，将红、绿、蓝三基色视频信号 R、G、B 分别送到相应的放大电路，用 R、G、B 与加亮信号的不同组合控制三束电子流的强弱。相应地荧光屏上的像素由能发出红、绿、蓝光的荧光粉小点组成，当一束（或两束、三束）电子流轰击对应的荧光粉小点时，屏上的像素便出现红、绿、蓝三基色之一或者由三基色合成的其他颜色，从而完成一帧画面的完整显示。

3. 同步控制

不论字符显示还是图形显示，都要求行、场扫描和视频信号的发送在时间上完全同步，即当电子束扫描到某字符或某像素的位置时，相应的视频信号必须同时输出。为此，在 CRT 显示器中设置了几个计数器，对显示器的主频脉冲进行分频，产生各种时序信号来控制对 VRAM 的访问，对 CRT 的水平扫描和垂直扫描，以及视频信号的产生等。

字符方式和图形方式下对计数器的设置是有区别的，下面分别进行详细讨论。

（1）字符/数字显示的同步控制

我们以常用微机的字符显示为例，说明几级同步计数器的设置、各级分频关系、以这几级计数器为基础所产生的控制信号及它们的作用。

在微机中最常用的显示规格为：每帧最大显示 25 行×80 列，字符点阵 7×9，字符区间 9×14。按照这种描述，我们将一排字符称为一行，每排字符由 9 条水平扫描线组成，排间间隔 5 条扫描线，我们对一条扫描线称为线，对一条线上的各像素称为点。在字符方式下设置了点计数器、字符计数器、线计数器和行计数器。

① 点计数分频(7+2)：1

设置点计数器的目的是为了提供下述控制信号：读显存，控制一个字符区间内的横向间隔消隐，对字符计数器计数。

每个字符点阵横向 7 个点，间隔 2 个点。点脉冲一方面控制视频信号产生像素，另一方面对点计数器进行计数控制。每计数 9 个点，计数器状态回到 0，完成一次计数循环，所以点计数器的分频关系为 9：1。

每次访问显存，读出一个显示字符的编码。以字符编码为高位地址，以扫描线号为低位地址，访问字符发生器 ROM，从中读出 7 位代码。由点脉冲控制时间，在屏幕的一根扫描线上依次显示 7 个像素（亮或暗）。所以，每当点计数器完成一次计数循环，就应访问一次显存，以读取下一次显示的字符的编码。同时向字符计数器提供一个计数脉冲，表示显示了一个字符的一线 7 点及 2 点间隔。

点计数器的状态表明：哪一段是字符点阵，应该显示；哪一段是字符横向间隔，应当消隐（即第1~7点显示，第8、9点消隐）。

② 字符计数分频 $(80+L):1$

设置字符计数器的目的是为了提供显存的低位地址信息、控制一条水平扫描线内的显示与消隐、向显示头提供水平同步信号、对线计数器计数等控制信号。

每显示一个字符，即点计数器一个计数循环后，字符计数器计数一次。X锯齿波的充电过程导致一次正程扫描，一行水平扫描线包含80个字符的显示区间。X锯齿波的放电过程导致一次回扫，而回扫期间应当消隐，不显示。在水平扫描中，左右边缘部分的线性度可能不好，只取中间线性度较好的区间提供显示，因此边缘部分也应当消隐。将回扫与边缘部分等消隐段折合成 L 个字符位置， L 的值与显示头技术有关。因此，字符计数器计数 $80+L$ 次之后，完成一次计数循环，分频关系为 $(80+L):1$ 。每完成一次计数循环，产生一次水平同步信号，启动又一次水平扫描，并使下一级的线计数器计数一次。

在一次计数循环中，有80个字符的连续区间是显示的，有相当于 L 个字符区间是消隐的。字符计数器的计数值可在一条水平扫描线中控制哪些段显示、哪些段消隐。

访问显存的地址，取决于该字符在屏幕上的显示位置（行号、列号）。因此，字符计数器提供的当前显示位置列号，可以作为产生缓存低位地址的依据。但是，在屏幕上的列号以80为模，显存地址采用二进制，所以实际计算地址值时，要先将字符的行、列位置编号做整体的进制转换。

③ 线计数分频 $(9+5):1$

设置线计数器的目的是为了提供如下控制信号：访问字符发生器ROM的低位地址，控制一行字符中哪些线显示及哪些线消隐，控制光标显示，对行计数器计数等。

每完成一次水平扫描（即字符计数器完成一个计数循环），则线计数器计数一次。一行字符占9条水平扫描线，然后是作为行间间隔的5条水平扫描线。所以，线计数器计数14次后，完成一个计数循环，分频关系14:1。线计数器完成一个计数循环后，对下一级的行计数器计数，启动新的一行字符显示。

如前所述，一行字符要分9次水平扫描才能完成显示。一个字符点阵信息占ROM的9个编址单元，每次访问ROM，只能读取一个字符点阵的一线像素信息。线计数器的计数值反映了当前的水平线序号，可作为ROM的低位地址。

在光栅进行的14次水平扫描中，第1~9线是显示段，第10~14线是消隐段。线计数器的计数值提供了对某一行的显示与消隐控制。在显示过程中一般都设置有光标，以低频闪烁的方式来指示下一个字符的显示位置。光标位于第10~14线的区间，线计数值用来控制光标显示位置的线号。

④ 行计数分频 $(25+M):1$

设置行计数器的目的是为了提供下述控制信号：提供显存的高位地址信息，向显示头提供垂直同步信号，控制一场显示过程中的显示段与消隐段。

每显示完一行字符，即线计数器完成一个计数循环，则行计数器计数一次。Y锯齿波的充电过程导致一次自上而下的正程扫描，可显示25行字符。回扫与线性度不好的边缘部分应当消隐，折合为 M 行， M 的值与显示头制造技术有关。因此，行计数器计数 $25+M$ 次之后，完成一个计数循环，分频关系 $(25+M):1$ 。相应地实现一场显示，发出一次垂直同步信号。

读显存时，地址码决定于显示字符的行列位置。行计数值决定了字符行号，可以作为产生高位地址的依据。由于行列号不是以 2^n 为模的，在计算实际地址时要进行进制转换。

在一场内，有 25 行显示段，另有折合为 M 行的消隐段，行计数器的计数值提供了一场中对显示与消隐的控制。

如果选择逐行扫描方式，则一场就是一帧。通常选取帧频为 60 Hz 以上，使视觉稳定，感觉不到闪烁。根据上述各级的计数分频关系，可由帧频推算出点脉冲频率及各级计数频率。这四级同步计数器与屏幕扫描之间的对应关系如图 6-18 所示。

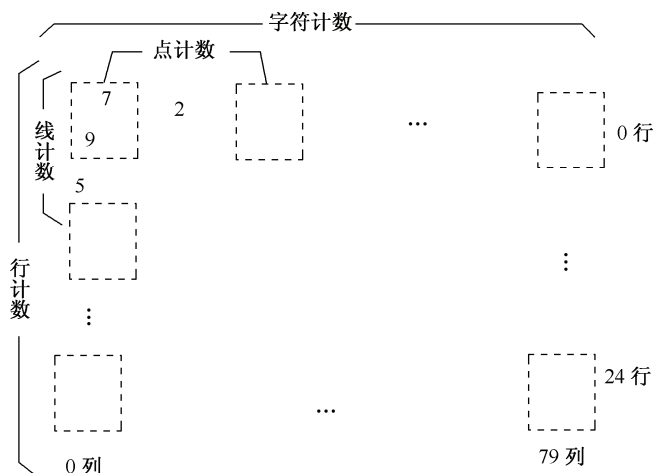


图 6-18 同步计数器与屏幕扫描

下面对字符显示的同步控制关系作一小结：

- ⊙ 点计数一个循环，访问显存一次以读取显示字符的编码，显存的地址根据行计数值与字符计数值决定。
- ⊙ 每读一次显存，就紧跟着读一次字符发生器 ROM，由显存读出的字符编码产生 ROM 的高位地址，线计数值决定 ROM 的低位地址。
- ⊙ 每次从 ROM 中读出显示字符的一线 7 位，由点脉冲控制逐位显示 7 点（亮或暗）。
- ⊙ 由于每条水平扫描线只能显示一行字符（可多达 80 个）的一线，所以上述访问显存、ROM 的过程需要重复 9 遍（每遍又要多次访问显存、ROM，以读取不同字符），才能显示完整的一行字符。
- ⊙ 字符计数循环一次，发送一次水平同步信号。
- ⊙ 行计数循环一次，发送一次垂直同步信号。

（2）图形/图像显示的同步控制

下面以某彩色图形显示为例，介绍 CRT 图形显示的同步控制原理。设该彩色图形显示器的分辨率为 640×480 ，可同时显示 16 种颜色。显存中存放着显示的图形点阵数据，由于计算机只能以二进制方式存放数据，每位只有两种状态（“0”或“1”）。对于单色显示，显存中的每一位对应画面上的一个像素点，该位为“1”即表示画面上的这一点是亮点。而对于彩色显示（如 16 种颜色），就需要用显存中的 4 位来定义一种颜色。在彩色图形显示器中经常采用彩色位平面的存储结构来表示颜色信息。每个彩色位平面由单一位组成，并表示屏幕上某个可以显示的颜色。例如分辨率为 640×480 ，每个位平面含有 640×480 位，即有 307 200 位的信息。由于要同时显示 16 种不同颜色，它就具有 4 个彩色位平面，故需要 1 228 800 位的显存，即 153 600 字节。所以，显存的总容量 = $640 \times 480 \times 4 \text{ bit} = 150 \text{ KB}$ 。它被分为 4 个位平面，每个位平面提供彩色代码中的一位，每个位平面的容量为 37.5 KB。

从屏幕显示角度，每行由 4 个位平面中的 80 字节来表示（ $640/8=80$ ）。屏幕上的一个彩色像素点，需要用来自 4 个位平面上每个位平面的相同位置的一个存储位表示。

根据上述的对应关系，可设计出显示器控制逻辑中的各计数器的同步计数分频关系，具体情况如图 6-19 所示。

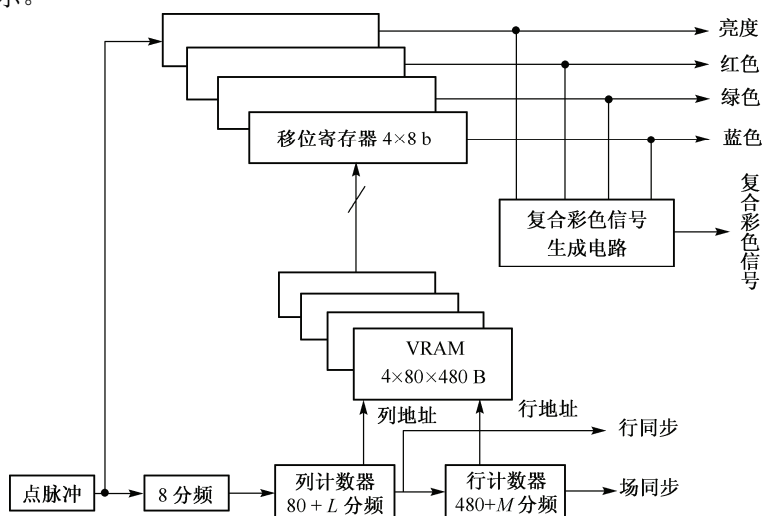


图 6-19 彩色 CRT 控制逻辑框图

① 点计数分频 8 : 1

图形以像素为单位，但在显存中以字节为单位按地址存储，即将一条水平线上自左向右，每 8 个点的代码作为 1 字节，存放在一个编址单元中。因此点脉冲经点计数器 8 分频之后产生字节脉冲，每发一次字节脉冲就访问一次显存，从 4 个位平面中各读出 1 字节（8 个点），送往移位寄存器，再串行输出形成亮度信号与红、绿、蓝三色信号，它们的组合决定了 16 色中的一种。若用于单色显示器，则将 4 位代码转换为 16 级亮度调制信号，用于控制像素的灰度。

② 字节计数分频(80+L) : 1

字节计数器又称为列计数器， $80+L$ 分频。计数值从 1 到 80，光栅从左向右扫描一行，正程显示 80 字节共 640 点。字节计数器所附加的 L 次计数，作为行线逆程回扫时间，逆程回扫时光栅扫描线应当消隐。

③ 线计数分频(480+M) : 1

线计数器又称行计数器， $480+M$ 分频。计数值从 1 到 480，对应于场正程扫描，显示 480 行（线）；附加 M 次计数，对应于场逆程回扫，逆程回扫应消隐。

行计数值与列计数值决定了屏幕当前显示位置（8 点一组），相应的显存地址为：行号 $\times 80$ +列号。按该地址同时访问 4 个位平面，取出 4 字节的图形代码。列计数一个循环，输出一个行扫描（水平）同步信号；行计数一个循环，输出一个场扫描（垂直）同步信号。这就使对 VRAM 的访问与 CRT 的扫描严格同步，能获得稳定的显示画面。

4. CRT 显示器的种类

（1）根据调控方式不同，可分为模拟调节、数字调节和 OSD 调节

模拟调节是在显示器外部设置一排排调节旋钮，转动这些旋钮可手动调节显示器的亮度、对比度等一些技术参数。由于模拟器件较多，故障的几率较大，而且可调节的内容极少，所以这种调节方式早已销声匿迹。

数字调节是在显示器内部加入专用微处理器，操作更精确，能够记忆显示模式，而且其使用的多是微触式按钮，这种调节方式寿命长，故障率低。

OSD (On-Screen Display) 即屏幕菜单式调节方式，严格地说也是数字调节方式的一种，能以量化的方式将调节直观地反映到屏幕上。目前的主流显示器大多采用这种调节方式。

(2) 按显像管种类的不同，可分为球面显像管、柱面显像管和纯平显像管

球面显像管的缺陷非常明显，在水平和垂直方向上都是弯曲的，边角失真现象严重，随着观察角度的改变，图像会发生倾斜，而且容易引起光线的反射，会降低对比度，对人眼的刺激较大。早期的黑白电视机，基本上都是球面显像管。

柱面显像管采用栅式荫罩板，在垂直方向上不存在任何弯曲，在水平方向上略有弧度。常见的柱面管可分为单枪三束和三枪三束管。

CRT 彩色显示器发展到后期，就出现了纯平显像管。纯平显像管在水平和垂直方向上均实现了真正的平面，失真、反光都被减到了最低限，使观看时的聚焦范围增大。

CRT 显示器在 20 世纪十分流行，随着技术的发展，逐渐被 LCD 等技术更先进的显示器代替。目前，在市场已很难再看到 CRT 显示器销售了，仅仅有少量的老旧电子设备还在继续使用 CRT 显示器，如示波器、心电监护仪等。

6.3.4 LCD 显示器

LCD 显示器即液晶显示器，原理上是通过控制液晶粒子的显示颜色来完成画面显示，它是继 CRT 显示器之后逐渐发展起来一种新型显示器，并逐渐取代了 CRT 显示器的地位，目前已成为了一类使用最广泛的显示器。

1. 液晶基本知识

1888 年，奥地利植物学家莱尼茨尔合成了一种奇怪的有机化合物，它有两个熔点。把它的固态晶体加热到 145°C 时，便熔成液体，只不过是浑浊的，而按照常理，之前发现的一切纯净物质熔化时应该是透明的。但是，如果接下来继续把这种有机化合物加热到 175°C ，它似乎又能再次熔化，进而从 145°C 时的浑浊状态变成 175°C 的清澈透明的液体。1889 年，德国物理学家莱曼在对同样的有机物进行研究时，发现了这种白浊物质具有多种弯曲性质。这种物质被认为是流动性结晶的一种，由此而取名为“液晶”(Liquid Crystal, LC)。

在自然界中，大部分材料像水一样，随温度的变化只呈现固态、液态和气态三种状态。而新发现的液晶是不同于固态、液态和气态的一种新的物质状态，它能在某个温度范围内兼有液体的流动性和晶体的光学性质，也叫做液晶相或中介相(物质的状态通常也称为“相”)。故液晶又称为物质的第四态。水和液晶的状态变化情况如图 6-20 所示。

2. 液晶的分类

液晶种类很多，如果根据材料的内部分子排列方式来分类，液晶可以分为 4 种：层状液晶(Sematic)、线状液晶(Nematic)、胆固醇液晶(Cholesteric)、碟状液晶(Disk)。图 6-21 是四种类型液晶的分子排列示意图。

(1) 层状液晶

层状液晶是由液晶棒状分子聚集一起，形成层状的结构。其每一层的分子的长轴方向相互平行，并且长轴方向相对于每一层平面呈垂直(或某一倾角)方向。由于结构非常近似于晶体，层状液晶又被称为“近晶相”。层状液晶的多个层面靠“键结”组合在一起。当温度变化的时候“键

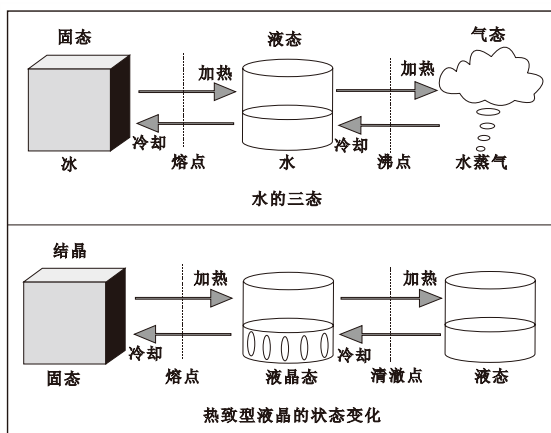


图 6-20 水与液晶的状态变化示意图

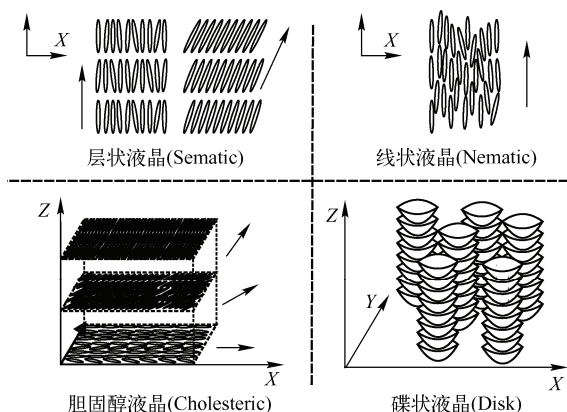


图 6-21 四种类型的液晶分子排列示意图

结”会容易断裂，所以层状液晶的层与层之间较容易滑动。但是每层内的分子“键结”比较强，不易被打断，就单层来看，分子排列不仅有序（分子长轴方向排列一致）而且黏性较大。在层状液晶中，每一层的液晶分子的长轴方向都具有相对于层状平面一定倾斜角度，而且通常层与层之间的分子倾角还会形成像螺旋的结构。

（2）线状液晶

线状液晶用希腊单词 **Nematic** 命名，义同英文单词 **Thread**。因为这种液晶分子的排列，看起来会有像丝线一样的形状。这种液晶分子在空间上具有一维的规则性排列，所有液晶分子呈长棒状，且长棒的长轴会选择某一特定方向，即“指向矢”（**Director**）。“指向矢”作为长棒状液晶分子的主轴，并且，所有的液晶分子都平行排列。线状液晶不像层状液晶那样具有分层结构，线状液晶与层状液晶相比，分子排列相对比较杂乱。

线状液晶的分子间黏度较小，所以较易流动（在长轴方向，分子较易自由运动）。在 TFT 液晶显示器的相关技术资料中常提到的 TN（**Twisted Nematic**）型液晶就是线状液晶。

（3）胆固醇液晶

胆固醇液晶大部分情况下是由胆固醇的衍生物所生成的，故名“胆固醇液晶”。然而，某些没有胆固醇结构的液晶也会具有这样的液晶相。如果把胆固醇液晶一层一层分开来看，层内分子排列很像线状液晶，但是从分层的垂直方向看，会发现它的“指向矢”会随着层与层的分子排列方向的不同而像螺旋状一样分布。胆固醇液晶的每一层分子由于其指向矢的不同，就会有光学和

电学特性的差异。

(4) 碟状液晶

碟状液晶也称为柱状液晶 (Discoid)。以单个的液晶分子来说, 它的形状像碟状 (Disk), 但是多个分子的排列会重叠排列, 像柱状。

如果从产生液晶的条件来看, 液晶又可以分为热致液晶 (Thermotropic LC) 和溶致液晶 (Lyotropic LC) 两大类。

(1) 热致液晶

某些有机物在加热时, 结晶晶格会被破坏掉, 有机物溶解, 进而形成的液晶, 这种液晶就是“热致液晶”。热致液晶的光电效应受控于温度条件。层状液晶与线状液晶一般多为热致液晶。到 2014 年, 液晶屏幕的液晶材料基本上都是热致型的液晶。

(2) 溶致液晶

某些有机物放在特定溶剂中, 结晶晶格会被溶剂破坏, 从而即可形成液晶, 这种液晶就是溶致液晶。溶致型液晶受控于溶剂的浓度条件。溶致液晶在溶剂中浓度很低时, 分子杂乱地分布于溶剂中, 形成“等方性”的溶液。但是当浓度升高大于某一临界浓度时, 由于分子已没有足够的空间来形成杂乱的分布, 部分分子开始聚集形成较规则的排列, 以减少空间的阻碍, 因此形成“异方性”的溶液。也就是说, 溶致型液晶的产生就是液晶分子在特定溶剂中达到某一临界浓度时, 便会形成液晶态。

溶致型的液晶有一个最好的例子, 就是水面上的肥皂泡。当肥皂泡漂在水面上时并不会立刻便成液态, 而在水中泡久了之后, 肥皂泡便会溶解为乳白状物质, 就是它的液晶态。

3. 液晶的光电特性

液晶是一种很特殊的物质, 以最常用的线状晶体为例, 其分子形状为长棒状, 长宽分别为 $1\sim 10\text{ nm}$ 。这种晶体在常规状态下, 分子长轴会选择某一特定方向作为主轴并相互平行排列, 这时当光线沿长轴方向从分子的一端入射时, 由于液晶分子在长轴方向具有和晶体分子一样良好的透光性, 光线几乎可以全部顺利从分子另一端透射出, 如图 6-22(a)所示。但是, 一旦给液晶施加某种电场, 则液晶分子的排列就会发生变化, 同时液晶的透光性也会随之发生改变, 进而会阻碍入射光线的通过, 如图 6-22(b)所示。液晶的这个特殊的光电特性常被用来制造 LCD 面板。

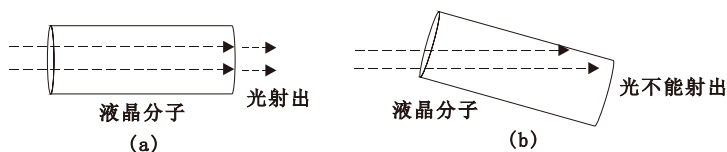


图 6-22 液晶分子透光原理图

常见的液晶显示器其形状外观如图 6-23 所示, 它比 CRT 显示器轻薄得多, 主要是由 LCD 面板模块、信号控制模块及逆变器和底座支架等组成, 其中 LCD 面板是液晶显示器最重要的核心模块, 也是 LCD 显示器与 CRT 显示器最本质差别所在。

1. LCD 面板构造及工作原理

LCD 屏幕是一块薄薄的面板。与 CRT 屏幕相比, LCD 屏幕的体积明显减小且更轻薄, 但其内部构造却比 CRT 更加复杂。如果对 LCD 面板进行切割和解剖, 然后观察其断面, 就会发现 LCD 面板是由很多层不同功能的薄片组成的, 其成分主要包括: 上偏光板、沟槽玻璃板 (上下配向膜)、液晶层、下偏光板和背光源板, 其剖面的分层结构如图 6-24 所示。



图 6-23 液晶显示器外观

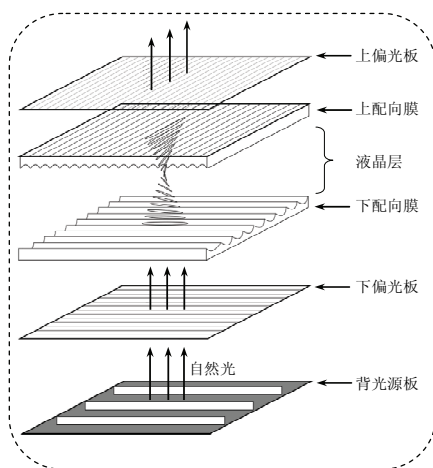


图 6-24 LCD 面板剖面结构

在图 6-24 中，LCD 面板的多层薄片从上至下排列，最上面是偏光板（上偏光板），依次向下分别是沟槽玻璃板或配向膜（上配向膜）、液晶层、下配向膜、下偏光板，最下面是由多个背光灯管组成的背光源板。其中，偏光板有上下两片、沟槽玻璃板（配向膜）也有上下两片（两片偏光板构造不同、两片玻璃板沟槽构造也不同），液晶层夹在沟槽玻璃板的中间。

从图 6-24 还可以看到 LCD 面板的成像过程：点亮背光源的灯管，背光源可以发出由下向上的自然光线，自然光首先可以透射穿过下偏光板、下沟槽玻璃板（下配向膜），来到液晶层。这时，液晶层在两旁矩阵排列的电线产生的电场信号的作用下按指定方向排列长棒形的液晶分子，液晶分子就可以控制光线按照指定方向偏转行进，这样，光线就可以从上沟槽玻璃板（上配向膜）透射出来而到达上偏光板，最后透过上偏光板射出的光线就会形成 LCD 屏幕上相应位置的像素“亮点”。反之，如果光线自下而上的行进过程中有任何一层薄片阻挡光线，光线最后无法到达屏幕的最上面，LCD 屏幕相应位置上的点就是像素“暗点”。

(1) 偏光板

由于光具有波粒二象性，所以光就有光波，其波长为 $380\sim 780\text{ nm}$ 。不同颜色的光波波长不同。在可见光中，红色波长最长，紫光波长最短。光波的波形是一种正弦振动波，自然光包含的光波振动面不只限于一个固定方向，而是在各方向上均匀分布。如果光的振动面只限于某一固定方向，这种光就叫做偏振光或线偏振光。自然光和偏振光对人眼的刺激效果相同，人眼无法鉴别射入眼睛的光是自然光还是偏振光。

能透过偏振光的透光板就叫偏光板。偏光板是特制的玻璃板或聚乙烯醇高分子化合物薄膜作为片基制作薄膜片。偏光板上有特制的密集的平行线条，这些线条让偏光板具备了这样的特性：只允许和偏光板上线条平行的光振动波通过，如图 6-25(a)所示，其他振动方向的光波全部被阻止通过，如图 6-25(b)所示。偏光板上能透光的方向称为偏光板的“透光轴”，也就是说，与透光轴平行的偏振光可以通过偏光板。

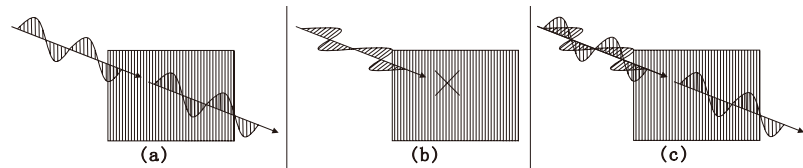


图 6-25 偏光板透光示意图

当自然光试图通过偏光板时，偏光板只允许自然光中振动面和偏光板透光轴平行的通过，如图 6-25(c)所示。可见，偏光板实际上的作用就是一个偏振光滤波器，专业领域一般也将它称为极化滤波器。

如果用两块透光轴相互垂直的偏光板重叠在一起，透过两块偏光板观看景物，这时景物自然光中振动面与第一块偏光板透光轴平行的偏振光就会穿透第一块偏光板，但是无法穿透第二块偏振轴垂直的偏光板，这就是 LCD 屏幕黑屏的状况。

(2) 液晶层

LCD 面板中的上下两块偏光板的透光轴是相互垂直的。如果没有液晶层作用的话，自然光是无法穿透两块偏光板透射形成图像的。但是，如果要保持这两块偏光板的位置不变，只要把通过第一块偏光板的水平偏振光扭曲 90° ，使之变成与偏振面垂直的偏振光后就可以穿透第二块偏光板了，基本原理如图 6-26 所示。“偏光板 A”和“偏光板 B”中间夹着一层液晶层，液晶层的主要作用就是控制由通过偏光板 A 之后的偏振光是扭曲还是不扭曲，从而决定偏振光能否继续穿透偏光板 B。

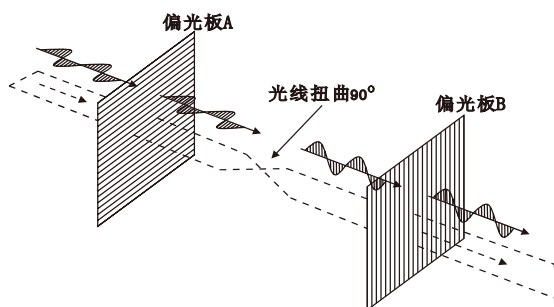


图 6-26 液晶层扭曲偏振光示意图

(3) 沟槽玻璃板与配向膜

在两片偏光板之间填充液晶必须要有一个“容器”来装载固定液晶。在如图 6-24 所示的 LCD 面板剖面结构中，一般用两片非常平整且透明度相当高的玻璃板充当液晶“容器”来夹住液晶。由于线状液晶分子的形状呈棒状，为了让液晶分子的排列更加整齐有序，在玻璃“容器”的内侧（临近液晶层的表面）上还有锯齿状的沟槽。沟槽就像小爪子，可以抓住身旁的液晶分子“棒”，使长棒形液晶分子沿着沟槽整齐排列。如果没有沟槽而只是光滑的平面，玻璃板就无法抓牢液晶分子，液晶分子的排列就会不整齐，光线透射液晶层时就会发生散射，从而导致 LCD 面板漏光。

在图 6-24 所示的面板剖面结构中，两片玻璃板的开槽方向相互垂直，靠近下面一块玻璃板的液晶分子排列方向和靠近上面一块玻璃板的液晶分子的排列方向也相互垂直。两片玻璃板之间的线状液晶分子在玻璃沟槽的抓附力、液晶分子间的黏附力和线状液晶分子同向排列的回弹力共同作用下，液晶层中的液晶分子就会呈现出规则的螺旋状排列结构。

在实际工业生产过程中，LCD 面板的这两块玻璃板“容器”内并没有沟槽，而是在上面涂一层比沟槽厚度更薄的化学制剂，这就是 LCD 的“配向膜”，其作用跟沟槽一样，可以让液晶分子从下到上呈螺旋状均匀排列在两块玻璃板之间。

不管是玻璃沟槽还是配向膜，在不加电场驱动的自然状态下，液晶层的分子从下到上呈螺旋状均匀排列，这种状态下的液晶分子会对两块偏光板的偏振光刚刚好扭曲 90° 。也就是说，在不加电场驱动的状态下，偏振光可以顺利穿透液晶层和两块偏光板，从而在 LCD 面板相应位置形成“亮点”，见图 6-24。

如果在两块玻璃板各设置一个电极，并施加一定的电压，就会在玻璃板中间的液晶分子的周边形成相应的电场。由于液晶分子对电场极其敏感，在电场的驱动作用下，液晶分子将打乱原来的螺旋状排列结构而进行重新的排列。这时液晶分子将不再受到配向膜的抓附力的控制，变为“竖立”排列状态，如图 6-27 所示。

在施加电场驱动的时候，偏振光将不会被液晶层扭曲，而是保持原来的偏振方向继续前行。偏振光到第二块偏光板后，由于偏振面和第二块偏光板的透光轴垂直，会被完全阻挡，从而在 LCD 面板相应位置形成“暗”点。

(4) 背光板

CRT 是通过电子枪发射电子束轰击屏幕上的荧光粉来产生亮点的，而 LCD 的液晶层本身并不发光，它只对偏振光进行扭曲控制。因此，为了产生图像，LCD 面板需要一个背光板来提供高亮且均匀分布的光源。背光源能提供自然光，一般采用两个或多个冷阴极荧光灯管作为背光源。背光源发出光线后，背光板的其他部件如导光板、限光板、扩散板、增亮膜则将背光均匀分布到 LCD 的各个区域，为液晶层提供分布均匀的入射光。

LCD 面板在通电工作的状态下，不管屏幕的点“亮”或“不亮”，LCD 面板中的背光灯管都是发光状态（除非 LCD 面板断电）。LCD 屏幕上之所以能呈现“暗点”（黑点），是因为在液晶层的作用下，背光被偏光板遮挡，但通常情况下，这种遮挡并不彻底，即 LCD 面板的暗点并不是绝对的没有任何光线透射的“纯黑点”，而是“黑灰点”。这也是 LCD 面板功耗较高的原因之一。

以上是 LCD 面板剖面结构及一个像素点的成像原理。要想知道 LCD 屏幕如何由“亮点”、“暗点”组合成一幅完整图像，还要了解 LCD 的矩阵驱动原理。

2. LCD 矩阵驱动方式

LCD 的成像原理是矩阵驱动原理，与 CRT “光栅扫描” 成像原理截然不同。

LCD 虽然也是把像素点进行排列组合来形成图像，但 LCD 像素点的排列组合方式完全不同于 CRT 的光栅扫描方式，LCD 是靠“矩阵电路”来驱动液晶层来呈现完整图像的。LCD 的“矩阵驱动电路”结构如图 6-28 所示。

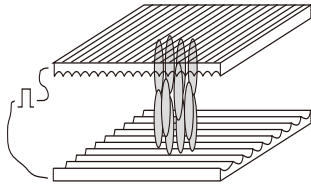


图 6-27 电场驱动下的液晶分子状态

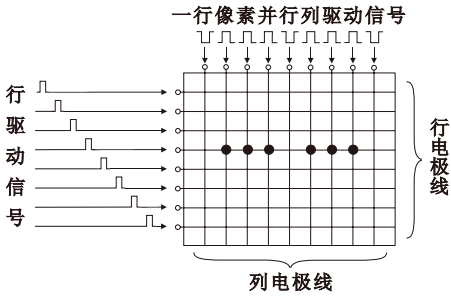


图 6-28 LCD 的矩阵驱动电路

这种“矩阵驱动”电路的结构特点是：在背光源一侧的偏光板和玻璃板之间安排一层矩阵驱动电极板。LCD 面板的分辨率与 CRT 分辨率表示规格相同，为“线数×点数”，矩阵电极板的行电极线数与 LCD 分辨率的“线数”相同，矩阵电极板的列电极线数与 LCD 分辨率的“点数”相同，以尺寸比例为 16：9、分辨率为 1920×1080 的全高清 LCD 面板为例，它的矩阵驱动电极板的水平行电极线有 1080 根，垂直列电极线有 1920 根。

行电极线和列电极线相互垂直，其交叉点位置刚好就是一个 LCD 面板像素点的位置。如果一个像素点位置对应的行电极线和列电极线同时施加电压，该像素点就会产生一个电场，在电场

驱动下，对应位置的液晶分子就会重新排列来控制偏振光的通过。

在 CRT 光栅扫描方式中，CRT 屏幕上图像/字符是从左至右、从上到下逐点扫描显示的。而 LCD 是通过水平的行电极线控制，在同一个时刻把一行视频信号的像素点同时显示在屏幕上的。从控制一行像素点的视频信号格式来看，CRT 是接收的串行格式的视频信号，并根据串行视频信号逐点扫描一行像素点的；LCD 是先通过时序转换电路把计算机发送过来的串行视频信号进行串并格式的转换，转换成一行像素对应的并行视频信号，再根据并行视频信号将一行的像素点同时显示出来，如图 6-29 所示。

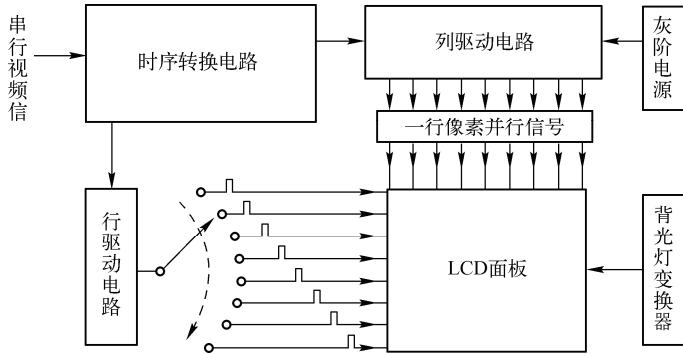


图 6-29 LCD 驱动系统的逻辑结构图

在 LCD 驱动系统中，计算机主机发送过来的串行视频信号首先经过一个“时序转换电路”转化为相应的“水平”（行）驱动信号和“垂直”（列）驱动信号，这些驱动信号再通过 LCD 的矩阵驱动电路“寻址”产生屏幕上相应像素点的驱动电场。屏幕上每个像素点都会有各自的电场驱动信号，在电场驱动下，液晶层中的液晶分子排列成将要显示图像的 shape（这时的液晶层就像经过感光的照相“底片”，然后“底片”经过背光源的光线照射，就能“显影”成为真实图像）。LCD 的这种成像方式就称为“矩阵驱动方式”。

3. LCD 的分类

根据液晶驱动方式的不同，LCD 可以分为被动矩阵式 LCD、主动矩阵式 LCD。

（1）被动矩阵式 LCD

被动矩阵式 LCD 在亮度及可视角方面受到较大的限制，响应速度也较慢。由于画面质量方面的问题，这种显示设备不利于发展为桌面型显示器。被动矩阵式 LCD 又可分为 TN-LCD（Twisted Nematic-LCD，扭曲向列 LCD）、STN-LCD（Super TN-LCD，超扭曲向列 LCD）和 DSTN-LCD（Double layer STN-LCD，双层超扭曲向列 LCD）三种。这三种被动矩阵式 LCD 的成像原理相同，只是液晶层的液晶分子排列扭曲角度不同。由于成像原理的缺陷，这三种被动矩阵式 LCD 中除 TN-LCD 还应被用做电子计算器等低端黑白屏幕外，其他两种几乎已经绝迹。

（2）主动矩阵式 LCD

主动矩阵式 LCD 是目前应用最广泛的一种 LCD。最有代表性的就是 TFT-LCD（Thin Film Transistor-LCD，薄膜晶体管 LCD）。TFT-LCD 面板在每个像素位置内建了一个 FET 晶体管（Field Effect Transistor，场效应晶体管），可控制光的穿透率，使面板亮度更明亮、色彩更丰富并提供更宽广的可视面积。TFT-LCD 是目前主流的计算机显示器。

TFT-LCD 面板结构中除了包含图 6-22 中的偏光板、配向膜、液晶层、背光板外，还有一块彩色滤光片。彩色滤光片的作用是使每个像素点都有 RGB 三个色彩块（原理同 CRT 显示器 RGB

色彩原理)。

在电路结构中, TFT-LCD 还有 FET 及驱动电路、背光控制模块等核心部件, 相互独立的 FET 晶体管驱动像素点发出彩色光, 就可以呈现高响应度、高亮度、高对比度的彩色图像信息。

与被动矩阵式 LCD 不同的是, 由于 TFT-LCD 的 FET 晶体管具有电容效应, 能够靠电容效应在一段时间内保持电场的稳定, 这样就可以将透光排列的液晶分子一直保持这种透光排列状态, 直到 FET 电极改变其排列方式为止。由于是通过晶体管的充电电容维持液晶分子的电场, 所以 TFT-LCD 的功耗相对被动式矩阵 LCD 非常低。

4. 平板显示器技术展望

以 LCD 为代表的平板显示器技术发展非常快速, 在 TFT-LCD 后又发展出 IPS-LCD (In-Plane Switching LCD, 平面转换 LCD)、S-LCD (Super LCD, 超级 LCD)。这两种 LCD 因为拥有更好的色彩还原、更宽广的视角、更薄的面板、更低的功耗而被广泛应用于各种平板电脑、智能手机以及 PC 显示器和家用大屏幕电视机。

目前还有一种不同于 LCD 显示方式的平板显示屏技术, 即 OLED (Organic Light-Emitting Diode, 有机发光二极管) 技术。由于显示方式的不同, OLED 相较于 LCD 有很多先天优势: OLED 具有自发光特性, 不需背光板, 因此 OLED 的面板较 LCD 面板更薄、功耗更低。

在 OLED 基础上, 以韩国三星公司为首, 已经研制出 AMOLED (Active Matrix OLED, 主动矩阵式 OLED) 面板, 并已经广泛应用于三星公司的多种平板电脑和智能手机上。随着技术的发展, 以 OLED 技术为基础的 AMOLED、Super AMOLED (超级 AMOLED) 面板产量将大幅提升、成本大幅降低, 将来极可能会取代 LCD 显示器的地位。

6.3.5 显示适配器及工作原理

显示规格取决于显示器控制器 (适配卡) 提供的显示规格 (也称为显示方式) 和显示头所能满足的分辨率要求这两方面。随着计算机技术的发展, 显示设备的不断升级换代是极为重要的, 它直接影响到用户所感受到的功能强弱。为此, 除了提高显示器本身的分辨率和清晰度外, 还应不断推出新的显示器适配卡。

IBM 公司为个人计算机先后主要推出了以下几种彩显适配卡: 彩色图形适配器 CGA (Color Graphics Adapter), 增强型图形适配器 EGA (Enhanced Graphics Adapter), 多色图形适配器 MCGA (Multiple Color Graphics Adapter), 视频图形阵列 VGA (Video Graphics Array), 扩展图形阵列 XGA (eXtended Graphics Array)。每种适配卡中允许编程选择几种显示规格。这些显示标准被称为 IBM PC 视频标准 (方式), 已为国际上普遍接受。各厂家在微机硬件开发和软件开发中, 都以与这些标准相兼容作为其产品性能的基本要求。

(1) CGA 接口

彩色图形适配器 CGA 是为 PC/XT 配置的, 有 7 种工作方式可供编程选择, 如表 6-4 所示。

(2) EGA 接口

增强型图形适配器 EGA 是为 PC/AT (286 机) 配置的。除兼容 CGA 的 7 种工作方式之外, 它新增加了 4 种工作方式, 如表 6-5 所示。

一些公司推出的汉化的彩显适配卡 CEGA, 在西文与图形显示方面与 EGA 兼容, 并具有汉字显示功能。还新增了两种显示规格: ① 字符方式, 以像素计算的分辨率可达 648×504 , 64 色中任选 16 色; ② 图形方式, 分辨率为 640×480 , 64 色中任选 16 色。

表 6-4 CGA 方式

| 方式 | 类型 | 分辨率 | 颜色 |
|----|----|-------------|-----|
| 0 | 字符 | 40 列×25 行 | 2 色 |
| 1 | 字符 | 40 列×25 行 | 4 色 |
| 2 | 字符 | 80 列×25 行 | 2 色 |
| 3 | 字符 | 80 列×25 行 | 4 色 |
| 4 | 图形 | 320 点×200 线 | 4 色 |
| 5 | 图形 | 320 点×200 线 | 2 色 |
| 6 | 图形 | 620 点×200 线 | 2 色 |

表 6-5 EGA 方式

| 方式 | 类型 | 分辨率 | 颜色 |
|-----|----|---------|----------|
| D | 图形 | 320×200 | 16 色 |
| E | 图形 | 640×200 | 16 色 |
| F | 图形 | 640×350 | 单色、4 种灰度 |
| 10H | 图形 | 640×350 | 16 色 |

(3) MCGA 接口

MCGA 是 IBM 为 PS/2 低档机型配置的多色图形适配器，与 CGA 兼容，并增加了两种方式，如表 6-6 所示。它的最高分辨率为 640×480，当分辨率较低（320×200）时，可同时显示 256 种颜色。MCGA 与 EGA 不兼容。

(4) VGA 方式

VGA（Video Graphics Array）接口由 15 针 D-Sub 组成，即 D 形三排 15 针插口方式，其中有一针未使用，对应连接使用的信号线空缺，但接触片是完整的，这种接口通常被简称为 D-15 型接口。

早期的 CRT 显示器因为设计制造上的技术原因，只能接受模拟信号输入，最基本的信号常常包含 R、G、B、H 和 V（分别对应红、绿、蓝、行和场）这 5 个分量，不管以何种类型的接口接入，其信号中至少包含以上这 5 个基本的信号分量。

VGA 接口除了前述 5 个必不可少的信号分量外，在 1996 年以后，还逐渐加入了 DDC 数据分量信号，通过它，可通过读取显示器中内置的 EPROM 存储芯片中记录的显示器品牌、型号、生产日期、序列号和各项指标参数等出厂信息内容，并根据显示器的这些基本信息，Windows 操作系统才能实现所要求的即插即用（Plug and Play，PNP）功能。

VGA 接口一般与 CGA 和 EGA 接口保持兼容，还增加了几种新的显示方式，如表 6-7 所示。

表 6-6 MCGA 方式

| 类型 | 分辨率 | 颜色数目 |
|----|---------|------|
| 图形 | 640×480 | 2 |
| 图形 | 320×200 | 256 |

表 6-7 VGA 增加的显示方式

| 方式 | 类型 | 分辨率 | 颜色数目 |
|-----|----|---------|------|
| 11H | 图形 | 640×480 | 2 |
| 12H | 图形 | 640×480 | 16 |
| 13H | 图形 | 320×200 | 256 |
| | 图形 | 800×600 | 256 |

通常，VGA 接口的技术指标包括分辨率、显示的颜色数量等都有了明显的提高，可用来显示高质量的、有真实感的图形，因此 VGA 已广泛地用于各种微机中。VGA 方式本身的规格正在继续提高，许多与 VGA 兼容且性能有各种改进的图形显示器也在不断涌现，有的快速图形显示器的分辨率甚至高达 3200×1024 像素。

(5) XGA 接口

XGA 是 IBM 公司继 VGA 之后开发出的扩展图形显示适配器。其中配置有协处理器，属于智能型适配器。XGA 可实现 VGA 的全部功能，但运行速度比 VGA 快，且其分辨率一般高达 1024×768 像素。

（6）DVI 接口

DVI（Digital Visual Interface，数字视频接口）是近年来随着数字化显示设备的发展而同步发展起来的一种新型显示接口标准。

普通 RGB 接口在显示过程中，先要在适配器中经过“数/模”转换，将数字信号转换为模拟信号输入到显示器。在数字化显示设备中，又要经“模/数”转换将模拟信号转换成数字信号再显示。在经过二次转换后，会造成信息丢失，对图像质量有一定影响。在 DVI 接口中直接以数字信号的方式将显示信息传输到显示器，可避免二次转换过程，因此从理论上讲，采用 DVI 接口的显示器的图像质量更好。

另外，DVI 接口实现了真正的即插即用（PNP）和热插拔（Hot-plugging 或 Hot Swap）功能，在连接过程中不需关闭计算机和显示设备。现在有很多 LCD（液晶）显示器都采用这种接口，但 CRT 显示器在被淘汰之前却很少使用 DVI 接口。

目前的 DVI 接口分为两种。一种是 DVI-D 接口，只能接收数字信号，接口上只有 3 排 8 列共 24 个针脚，其中右上角的一个针脚为空，不兼容模拟信号。另一种是 DVI-I 接口，可同时兼容模拟和数字信号。兼容模拟信号并不意味着 RGB 模拟信号的 D-15 接口可以直接连接到 DVI-I 接口上，而是必须通过一个转换接头才能使用，一般采用这种接口的显示适配器都会带有相匹配的转换接头。

考虑到兼容性问题，目前的显示适配器一般会采用 DVI-I 接口，这样可以通过转换接头连接到普通的 VGA 接口。带有 DVI 接口的显示器一般使用 DVI-D 接口，因为这样的显示器一般也带有 VGA 接口，所以不需要带有模拟信号的 DVI-I 接口。当然也有少数例外，有些显示器只有 DVI-I 接口而没有 VGA 接口。

（7）HDMI 接口

HDMI（High Definition Multimedia，高清晰度多媒体接口）接口的音频/视频采用同一电缆进行传输，可以提供高达 5 Gbps 的数据传输带宽，可以传输无压缩的音频信号及高分辨率视频信号。同时，不需在信号传输前进行“数/模”或者“模/数”转换，可以保证最高质量的影音信号传输。此外，HDMI 接口支持 HDCP 协议，为收看有版权的高清视频奠定了基础。

（8）DisplayPort 接口

DisplayPort 也是一种高清数字显示接口标准，这种标准最早在 2006 年 5 月由视频电子标准协会（VESA）确定了 1.0 版标准，并在半年后升级到 1.1 版。DisplayPort 既可以连接计算机与显示器，也可以连接计算机与家庭影院，它是一种针对所有显示设备（包括内部和外部接口）的开放标准，甚至可以全面取代 DVI 和 VGA 接口。

从性能上讲，DisplayPort 1.1 最大支持 10.8 Gbps 的传输带宽，最新的 HDMI 1.3 标准也仅支持 10.2 Gbps 的带宽；DisplayPort 可支持 WQXGA+（2560×1600）、QXGA（2048×1536）等分辨率及 30/36 bit（每原色 10/12 bit）的色深，1920×1200 分辨率的色彩支持到了 120/24 bit，超高的带宽和分辨率完全足以适应显示设备的发展。

与 HDMI 一样，DisplayPort 也允许音频与视频信号共用一条线缆传输，支持多种高质量数字音频。但比 HDMI 更先进的是，DisplayPort 在一条线缆上还可实现更多的功能。在四条主传输通道之外，DisplayPort 提供了一条功能强大的辅助通道。该辅助通道的传输带宽为 1 Mbps，最高延迟仅为 500 μs，可以直接作为语音、视频等低带宽数据的传输通道，也可用于无延迟的游戏控制。

DisplayPort 的外接型接头有两种：一种是标准型，类似 USB、HDMI 等接头；另一种是 Mini 型，由 Apple 公司开发，主要针对连接面积有限的应用，如超薄笔记型电脑。这两种接头都有 20

针，但迷你接头的宽度约是全尺寸的一半，两种接头的最长外接距离都可以达到 15 m。除实现设备与设备之间的连接外，DisplayPort 还可用作设备内部的接口，甚至芯片与芯片之间的数据接口。

(9) ADC 接口

除了以上介绍的几种显示设备常见的接口外，还有一种 ADC (Apple Display Connector) 接口，也是 Apple 公司为显示器开发的一种专用显示器接口。

从用户的角度看，ADC 接口集成了 DVI-D 接口、USB 接口和 DC (直流电源) 接口，其最大特点就是数据线和电源线集成在一起，显示器只需一根线就能满足 Apple 系列计算机清爽时尚的风格。目前，Apple 公司生产的系列显示器产品，都广泛采用了 ADC 接口。

在前述介绍的几种适配器接口中，VGA、DVI、HDMI 和 DisplayProt 等接口的应用目前比较广泛。其他接口标准，如 CGA、EGA 和 XGA 等，有的因为技术落后已被淘汰，有的因为各厂商的支持度不高而没有大规模发展起来。

下面以个人计算机中的一种显示器适配卡为例，说明显示器控制器及接口的基本逻辑组成与工作原理。图 6-30 是 IBM PC 机的一种单色显示器适配卡，从中可以了解 CRT 显示器硬件逻辑组成原理。

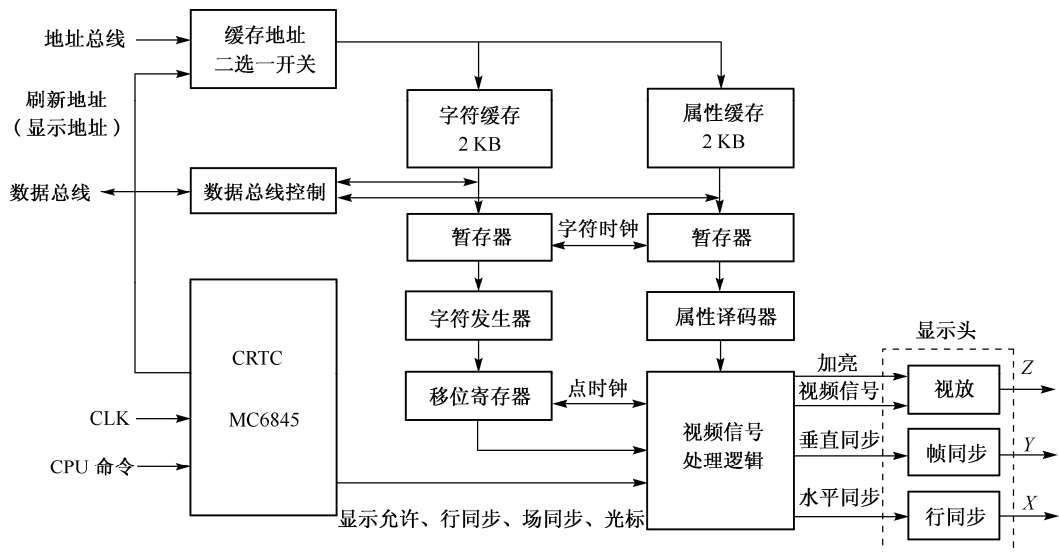


图 6-30 显示器控制器逻辑框图

1. 逻辑组成

(1) CRTC

该适配卡的核心是一种 CRT 控制器芯片 MC6845，包含与 CPU 连接的接口控制逻辑、18 个数据寄存器、除了点计数器外的其余 3 种同步计数器。这 18 个寄存器为：水平定时发生器 $R_0 \sim R_3$ ，垂直定时发生器 $R_4 \sim R_7$ ，光栅扫描方式寄存器 R_8 ，光栅地址发生器 R_9 ，光标逻辑 R_{10} 、 R_{11} 、 R_{14} 、 R_{15} ，显存刷新地址 R_{12} 、 R_{13} ，光笔逻辑 R_{16} 、 R_{17} 。

这些寄存器可由 CPU 编程设置，根据它们的内容来决定同步计数器的计数分频关系，并产生显存地址和各种控制信号。以水平定时发生器为例， R_0 中存放水平显示的总字符数，其中包含回扫等折舍值 L ； R_1 中存放每行可显示的最大字符数； R_2 中存放的信息决定以字符为单位的水平同步扫描位置，即字符计数器计数到何值时应发送水平同步信号； R_3 中存放的信息决定以字符为单位的水平同步扫描的脉冲信号宽度。

(2) 显存

显存分为字符缓存、属性缓存两个存储体，统一编址。当 CPU 向显存写入显示内容时，由 CPU 向适配器送出其地址。当适配卡从显存中读取显示信息时，由 CRTC 产生地址。因此，适配卡中设置缓存地址切换开关，从两种地址中选择一个送往显存。

(3) 视频信号处理逻辑

送入视频信号处理逻辑的有反映字符点阵的代码（以串行方式逐位送入）、属性代码、水平同步、垂直同步、光标信息。综合处理后产生送往显示头的信号有视频信号、加亮信号、行同步、场同步。

2. 工作过程举例

(1) CRTC 初始化

以微机为例，在启动显示系统工作之前，需先对 CRTC 进行初始化，并检查缓存容量。可以

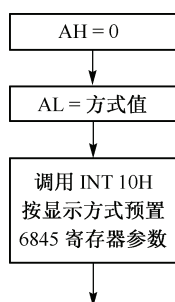


图 6-31 CRTC 初始化流程

通过软中断“INT 10H”调用 BIOS 中的显示器驱动程序，对 MC6845 内部各寄存器预置内容，即 CRTC 初始化。在执行“INT 10H”前，先向 8088 CPU 的 AH 寄存器写入功能码 0，表示“INT 10H”将完成设置显示方式的操作。同时，向 AL 寄存器写入方式码。例如，AL=0 表示显示方式为 25 行×40 列的黑白字符方式，AL=1 表示 25 行×40 列的彩色字符方式，AL=2 表示 25 行×80 列的黑白字符方式，AL=3 表示 25 行×80 列的彩色字符方式，AL=4 表示 320×200 彩色图像方式等。执行 INT 10H 的系统功能调用时，会按 AL 值建立显示规格。初始化流程如图 6-31 所示。

(2) CPU 向显示缓存写入显示内容

在水平回扫和垂直回扫等消隐期间，CPU 可用中断方式将显示内容（如字符编码与属性代码）送入适配卡的显存。此时，由 CPU 提供地址和显示信息。由于是利用回扫期间，不影响正程扫描时的访问缓存。

(3) CRTC 同步访存与扫描控制

在正程扫描期间，MC6845 内部的符合电路，不断地将有关寄存器内容与各计数器的值进行比较，产生有关控制信号，如读显存、地址、行同步、场同步、显示允许、光标等信号。周而复始地从显存中读出字符编码，经字符发生器转换为字符点阵代码，串行送入视频信号处理逻辑。视频信号处理逻辑输出视频信号，送显示头，经视频放大后送至 CRT 控制栅极 Z，产生像素。6845 产生的行同步信号与场同步信号，经视频信号处理逻辑产生水平同步信号与垂直同步信号，送显示头。再经行同步电路产生 X 锯齿波，经帧（场）同步电路产生 Y 锯齿波，由此控制屏幕扫描。

6.4 打印设备及接口

6.4.1 概述

打印设备是计算机的重要输出设备之一，能将计算机的处理结果以字符和图形等人们所能直观识别的形式打印在相关介质上，作为硬拷贝长期保存。为适应计算机飞速发展的需要，打印设备已从传统的机械式打印发展到新型的电子式打印，从逐字顺序打印发展到成行打印，从窄行打印（每行打印几十个字符）发展到宽行打印（每行打印上百个字符），并继续朝着不断提高打印

速度、降低噪声、提高打印清晰度和彩色打印等方向发展。

打印设备品种繁多，按一行字的形成方式、工作时是否有击打动作和打印字符的结构及采用的技术等，可将打印设备分为如下类型。

（1）串行打印和并行打印

按数据传输方式的不同，打印设备可分为串行打印机和并行打印机两类。

串行打印时，一行字符按先后顺序逐字打印，速度较慢，一般用“字符/秒”来衡量其打印速度。

并行打印也称为行式打印，一次同时打印一行中的某种相同字符，打印输出过程是逐行进行的。行式打印常常需要设置缓冲存储器，用来暂时存放准备打印的一行字符。这种方式打印速度比串行方式快，常用“行/秒”或“行/分”作为速度单位。

（2）击打式打印和非击打式打印

按印字方法的不同，打印设备可分为击打式打印机和非击打式打印机两种。

击打式打印机是通过字锤或字模的机械运动推动字符击打色带，使色带与纸接触，从而在纸上印出字符。当色带与纸接触的瞬间，若字符和纸处于相对静止状态，则称为“静印”方式。有的打印机（如快速宽行打印机）多采用“飞印”方式印字，以提高打印速度，字符被字轮带动高速旋转，在击打的瞬间，字符和纸张之间有微小的相对位移，故称为“飞印”或“飞打”。

非击打式打印机具有打印速度快、噪声小（或者无噪声）、印刷质量好等优点，它们通过电子、化学、激光等非机械方式来印字。例如，激光打印机、磁打印机等，利用激光或磁场先在字符载体上形成潜像，然后转印在普通纸上形成字符或图形；喷墨打印机不通过中间字符载体，由电荷控制直接在普通纸上印字；静电打印机及热敏、电敏式打印机等通过静电、热、化学反应等作用，在特殊纸上印出图像。

（3）全形字打印和点阵打印

按字符产生方式来划分，打印设备有全形字和点阵打印两类。全形字是将字模（活字）装在链、球、盘或鼓上，用打印锤击打字模将字符印在纸上（正印），或者打印锤击打纸和色带，使纸和色带压向字模实现印字（反印）。字模型用在击打式打印机中，印出的字迹清晰，但组字不灵活，且不能打印图形、汉字等图像。

点阵式打印机不用字模产生字符，而是将字符以点阵形式存放在字符发生器中。印字时，用取出的点阵代码控制在纸上打印出字符的点阵图形。常用的字符点阵为 5×7、7×9、9×9，汉字点阵为 24×24。点阵式打印机组字灵活，可以打印各种字符、汉字、图形、表格等，且打印质量也很高。针式打印机及所有非击打式打印机等均采用点阵型。

（4）针式、喷墨式和激光打印机

针式打印机是通过打印头中的 24（或 9）根针击打复写纸，从而形成字体。针式打印机在打印机历史上曾经占据重要地位，在很长的一段时间内流行不衰，具有打印成本低、易用性好等优点，特别适合多联票据的打印。针式打印机也有打印质量差、噪声大以及打印速度慢等缺点，所以只在银行、超市等需要打印票据的场合在使用。

喷墨打印机是在针式打印机之后发展起来的，采用非打击的工作方式，比较突出的优点有体积小、操作简单方便和打印噪音低，如使用专用纸张可以打印高质量照片。喷墨打印机按打印头的工作方式可以分为压电喷墨技术和热喷墨技术两种，按照喷墨的材料性质又可以分为水质料、固态油墨和液态油墨等。

激光打印的原理是将待打印页面转换成电信号控制发射激光，经过反射后让感光鼓感光，并将感光鼓上的碳粉转移到纸上的对应位置并进行高温加热融化固定，从而完成打印。激光打印机

有黑白打印和彩色打印两种，能提供打印质量更高、快速速度更快及打印成本更低的打印，有望全面替代喷墨打印机。

本节着重介绍在计算机系统中早期曾广泛使用的串行点阵针式打印机，并对它的结构和工作原理进行较为详细的讨论，然后介绍目前广泛应用的激光打印机、喷墨打印机的基本原理。

6.4.2 打印机的性能指标

与打印机相关的性能指标主要有打印分辨率、打印速度、打印幅面、接口方式、缓冲区的大小和打印介质类型等。

(1) 打印分辨率 (dpi)

打印机的打印质量是指打印出的字符的清晰度和美观程度，常用打印分辨率来表示，单位为每英寸打印的点数 (dot per inch, dpi)。针式打印机的分辨率一般为 180 dpi，喷墨打印机的分辨率一般为 720 dpi，激光打印机的分辨率一般为 300 dpi 或 600 dpi 以上，甚至可高达 1200 dpi。低档的精密照排机约为 700~2000 dpi，高档的则可达 2000~3000 dpi。

(2) 打印速度

打印速度可分为串式、行式和页式打印速度。如前所述，串式打印机的打印速度用每秒钟的字符数 (cps) 来表示，行式打印机用每分钟打印的行数 (lpm) 来表示，页式打印机用每分钟打印的页数 (ppm) 来表示。表 6-8 列出了高、中、低速打印机的打印速度。

表 6-8 打印机的打印速度

| 打印机类型 | 低速 | 中速 | 高速 |
|-------|------------|-------------|------------|
| 串式 | 小于 30 cps | 30~200 cps | 大于 200 cps |
| 行式 | 小于 150 lpm | 150~600 lpm | 大于 600 lpm |
| 页式 | 小于 10 ppm | 10~30 ppm | 大于 30 ppm |

(3) 打印幅面

打印幅面是衡量打印机输出文图页面大小的指标。

针式打印机中一般给出行宽，用一行中能打印多少字符 (字符/行或列/行) 表示。常用的打印机有 80 列和 132/136 列两种。

激光打印机常用单页纸的规格表示，可以分为 A3、A4 或 A5 等幅面打印机。打印机的打印幅面越大，打印的范围越大。

喷墨打印机也常用单页纸的规格表示。通常，喷墨打印机的打印幅面为 A3 或 A4 大小。有的喷墨打印机也使用行宽表示打印幅面。

(4) 接口方式

打印机的接口在早期大多数都配置标准的并行接口，一般称为 centronics 接口，也称为 IEEE1284，其最高传输速率可达 16Mbps，其他标准接口一般作为附件而需另外购置。

有的打印机也采用串行接口，其特点是传输速率慢但传输距离比并口更长。现在的台式计算机一般有两个串行口 COM1 和 COM2，通常 COM1 使用 9 针 D 形 RS-232 接口，而 COM2 使用老式 25 针 RS-422 接口 (目前已很少使用)。

目前的打印机一般支持 USB 接口。普通的 USB 接口能提供 12 Mbps 的传输速率，相比早期的并口其速率提高 10 倍以上，使打印文件的传输时间大大缩减。如采用 USB 2.0 接口标准，则能进一步将传输速率提高到 480 Mbps，又能进一步减少打印文件的传输时间。

(5) 缓冲区

打印机的缓冲区相当于计算机的内存，单位为 KB 或 MB。最简单的缓冲区只能存放一行打印信息，通常小于 256 字节，主机只能发送一行信息给打印机。当这一行信息打印完后，即清除缓冲区的信息，并告诉主机“缓冲区空”，主机将再发送新的信息给打印机，如此反复直到所有信息打印完毕为止。

在主机 CPU 不断升级的情况下，为了解决计算机和打印机速度的差异，必须扩大打印机的缓冲区。目前，24 针的针式打印机的缓冲区一般为 64 KB，也有高达 128 KB 的；喷墨打印机一般为 64 KB 或 128 KB；激光打印机的缓冲区容量一般为 4~8 MB，高端机型有的可高达 256 MB。缓冲区越大，一次输入数据就越多，打印机处理打印任务所需的时间就越长，因此与主机的通信次数就可以减少，从而提高主机的效率。

6.4.3 点阵针式打印机

点阵针式打印机（简称针打）是一种串行击打式打印机，与其他击打式打印机不同，它不是通过击打字模印字，而是靠若干根钢针在字符点阵代码的控制下击打色带和纸，在纸面上印出与点阵代码相应的字符图案。它的打印速度比其他串行打印机速度快，目前达 180~300 字符/秒，有的甚至高达 600 字符/秒。针打的字库容量可以很大，这就为打印图形和汉字提供了条件。打印针的工作频率较高，驱动打印针的能量又很小，因此打印时所发出的噪声容易被吸收和阻尼，使噪声降低。另外，针式打印的结构简单，使用方便，成本也低，所以获得了广泛的应用，特别在微机系统中已成为必不可少的硬拷贝输出装置。针打的主要缺点是印出的字迹不如字模型打印机清晰，但近年来采用了 NLQ（Near Letter Quality，准铅字质量）技术，使印字质量大大提高，几乎接近字模型。

1. 打印方式

与显示器的显示方式相类似，针式打印机也有两种打印方式。一种是文本字符方式，根据待打印字符的编码从字符发生器依次取出字符的各列点阵数据，控制钢针在纸上打印出一个一个的字符。与字符显示方式不同的是，字符发生器按列组织字符点阵代码而不是按行组织；并且点阵数据由 ROM 取出后，不经过并串转换而直接送往打印头。另一种打印方式是点图形方式，将图形的点数据存入缓冲存储器 RAM 中，打印时从 RAM 取出点数据直接送打印头，驱动钢针打印出图形或汉字。

2. 基本结构

针式打印机主要由打印头、小车横移机构、走纸机构、色带机构、保护装置和控制系统几部分组成。其中，打印头和控制系统是针打的核心。

(1) 打印头

打印机的关键部件是打印头，由电磁铁、打印针、导向部件和复位部件组成，如图 6-32 所示。每根打印针通过各自的导向管、复位弹簧和衔铁与一个电磁铁相连。打印时，在相应电磁铁线圈中通入电流，产生磁场，使衔铁杠杆在磁场作用下被吸合而运动，压缩弹簧，推动打印针沿导向管并通过导板击打色带，将色带和纸压向滚筒，从而在纸上印出一个小点。断开电流时，磁场消失，衔铁杠杆被释放，使被压缩的弹簧产生弹力，将打印针拉回起始位置。

电磁铁和打印针一般排成圆周形，打印针通过导向部件后，前端排成一行（9 针，如 FX-80、100）或互相错位排成两行（24 针，如 M2024、TH3070 等），如图 6-33 所示。24 根针排成奇数

针和偶数针各一列，呈锯齿状。打印时，先打印一列针，打印头移动一定距离后再打印另一列针，两次击打打印完字符的同一列点阵。

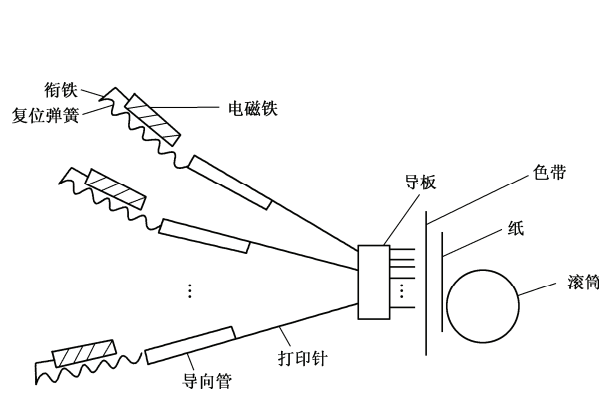


图 6-32 打印头结构

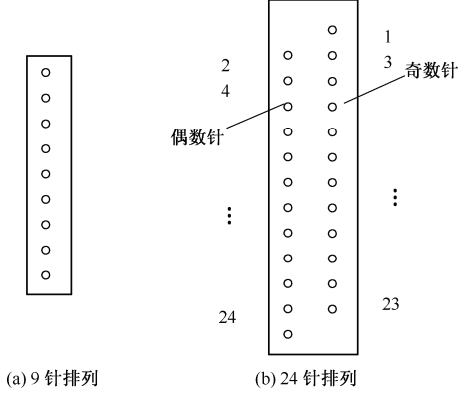


图 6-33 打印针的排列形式

打印头是靠电磁铁线圈中电流的有无来驱动打印针移动的，有螺管式、拍合式、储能释放式等驱动方式。

以目前仍在一些特殊行业（如银行的票据单、超市的购物单等）使用的储能释放式为例，它是利用永久磁铁的吸力使复位弹簧储存一定能量，作为推动针运动的驱动源。不打印时，由永久磁铁吸住衔铁、弹簧和针，弹簧被压缩而储备能量，使打印针处于准备击发的状态。打印时，在另一个线圈（释放线圈）中通入电流，该线圈所产生的磁场方向与永久磁铁的磁场方向相反，因而削弱甚至抵消永磁吸力，将衔铁释放，在所存储的弹性能的作用下，由簧片推动针打印。当打印动作完成后，切断释放线圈的电流，永久磁铁又恢复吸力，重新吸住衔铁，使弹簧再次储能，并迫使打印针很快回到起始位置。这种驱动方式使针的打印速度较快，可达 600 字符/秒。

还有一些驱动方式，这里不再一一介绍。

（2）水平横移机构

打印头装在小车上，由步进电动机或直流电动机驱动作水平方向的往返运动。伺服电动机上装有位置检测编码盘，当小车移动时，编码盘与之同步旋转。盘上刻有光栅，通过光电转换检测小车的位置。每次打印字符的一列点阵时，由小车将打印头移动到打印位置，打印头根据列点阵的 1、0 代码控制相应的打印针运动或不运动。打印完一列后，小车再拖动打印头到第二列位置继续打印。

（3）走纸机构

在打印一行字符的过程中纸是不动的，打印完一行后纸走动一格，以便打印下一行字符，因此走纸动作是间歇的。通常由步进电动机驱动滚筒按顺时针方向旋转，滚筒每旋转 36°，纸将通过摩擦方式或针牵引方式向前移动一步。

（4）色带机构

色带直接受打印针击打，若打印时色带不动，固定在一个位置打印，会很快被打破。但色带不能运动太快，否则会使印字模糊。所以，一般用电动机拖动色带按一定速度移动，并将色带头尾相接，装在色带盒内。当小车正反运动时，色带轴带动色带沿一个方向旋转，可使色带均匀使用，不致因某段色带使用太频繁而过早损坏。

（5）保护装置

当出现卡纸、纸歪斜或纸用完等情况时，保护装置均发出报警信号，如指示灯闪烁或发出响

声，以便操作人员及时进行处理。

3. 打印机控制器举例

现在的打印机都广泛采用了微处理器芯片对打印过程进行控制，有单 CPU 控制和双 CPU 控制两种方式。例如，M2024 打印机控制器便采用主、从双 CPU 方式，整个控制电路分布在接口板和逻辑板上，如图 6-34 所示。其中，位于接口板上的输入端口、输出端口和控制电路分别接收输入数据和控制命令，并返回输出状态。8 KB 的 RAM 作为行缓冲器和程序工作区，用行缓冲器事先存放待打印的一行字符信息。

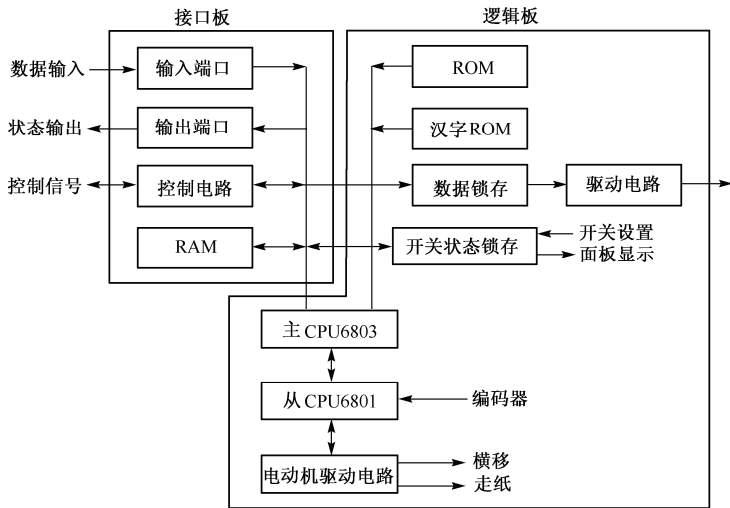


图 6-34 打印机控制器框图

位于逻辑板上的 8 KB ROM 划分一部分空间作为 ASCII 字符发生器，存放字符点阵信息，另一部分空间用来存放打印主控程序。汉字 ROM 作为选件，如果不配置该选件，则汉字库可由软盘提供。数据锁存器接收一列点阵信息（该列信息先由 ROM 字符发生器送 CPU 进行某些处理，再送往数据锁存器），提供给驱动电路控制打印头上的针的动作。主 CPU 通过执行主控程序对输入数据进行分析，接收主机送来的控制命令，控制打印机各部分协调操作，完成打印功能，并将打印机状态返回主机和面板指示灯。从 CPU 则根据主 CPU 的命令，通过电动机驱动电路控制直流电动机和步进电动机的动作，完成小车横移、走纸等辅助打印操作，并向主 CPU 回送小车运行状态的标志。

4. 字符打印过程举例

字符打印过程（以微机为例）由主控程序控制进行，主要包含打印机初始化、接收代码和打印处理三个阶段。

（1）打印机初始化

有两种方式可使打印机进入初始化状态。一种方式是加电后，直接由硬件电路使主控程序进入初始化阶段。另一种方式是在联机状态下，由主机通过软中断 INT 17H 调用 BIOS 提供的打印机驱动程序，并置功能号 AH = 1，向打印机发初始化命令，使主控程序也进入初始化阶段。

打印机初始化操作包括设置输入/输出端口工作状态、检查 RAM 是否正常、检测打印机某些关键部位有无异常并给出相应标志等。初始化流程如图 6-35 所示。

(2) 接收代码

打印机初始化操作完成之后, 向主机提出中断请求, 进入接收代码阶段。主机置 $AH=0$, 向打印机输入数据。主机送入打印机的代码有两种类型, 一类是要打印的字符或图形, 另一类是不打印的控制字符, 如回车 (CR)、换行 (LF)、换页 (FF) 等功能码。因此打印机控制器要识别这两类代码, 以便做出不同的处理。

图 6-36 给出了接收代码流程。不管是打印字符还是控制字符, 主机均以 ASCII 码形式将字符由 AL 送入打印机控制器输入端口。主 CPU 首先判断该字符代码是否为功能码, 如果是功能码, 则转相应的功能码处理程序; 如果是打印字符, 且打印机工作在字符方式下, 则从 ROM 字符发生器中选出相应的字符点阵信息, 并存入打印码缓冲区; 如果打印机工作在图形方式下, 那么接收的代码就是打印码, 不经 ROM 转换而直接存入打印码缓冲区。打印机以中断方式继续从主机接收代码, 并作相应转换, 直至装满一行打印码缓冲区, 或接收的代码是打印命令, 于是主控程序便转入打印处理阶段。

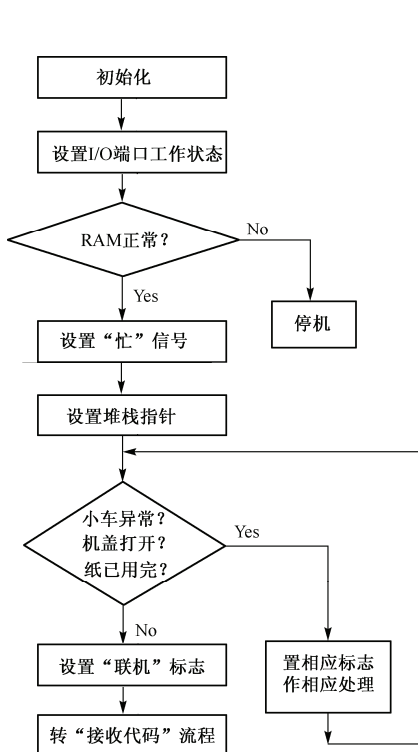


图 6-35 打印机初始化流程

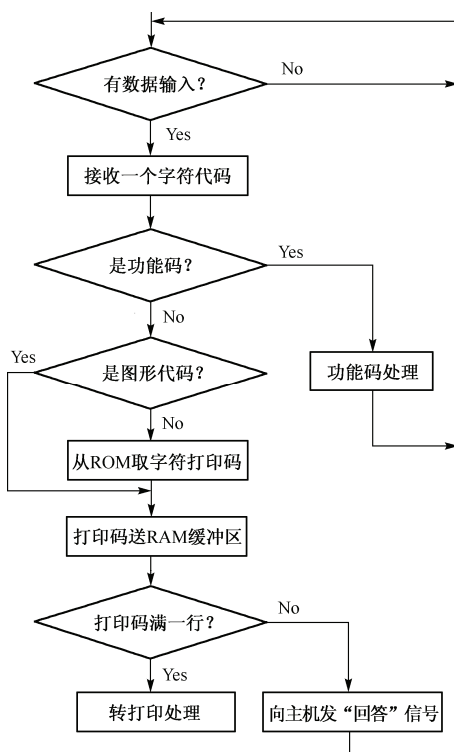


图 6-36 打印机接收代码流程

(3) 打印处理

一行打印码往往按字符分为若干个连续区间, 每个字符区间的打印码被连续打印, 字符和字符之间的空格不打印, 只需走车。

打印处理的基本流程如图 6-37 所示。在打印处理阶段, 主控程序首先确定第一个连续区间的打印码在缓存的头、尾地址指针, 并把该区间需要打印的全部数据和有关的打印命令及参数发送给 CPU, 确定是正向打印或反向打印, 并给出相应标志, 然后进行打印中断处理。若采用自左向右的正向打印, 则在打印一列点时, 先驱动奇数针打印, 待偶数针随打印头右移到同一列置时, 再驱动偶数针打印; 若自右向左反向打印, 则先驱动偶数针打印, 再驱动奇数针打印。当第一个

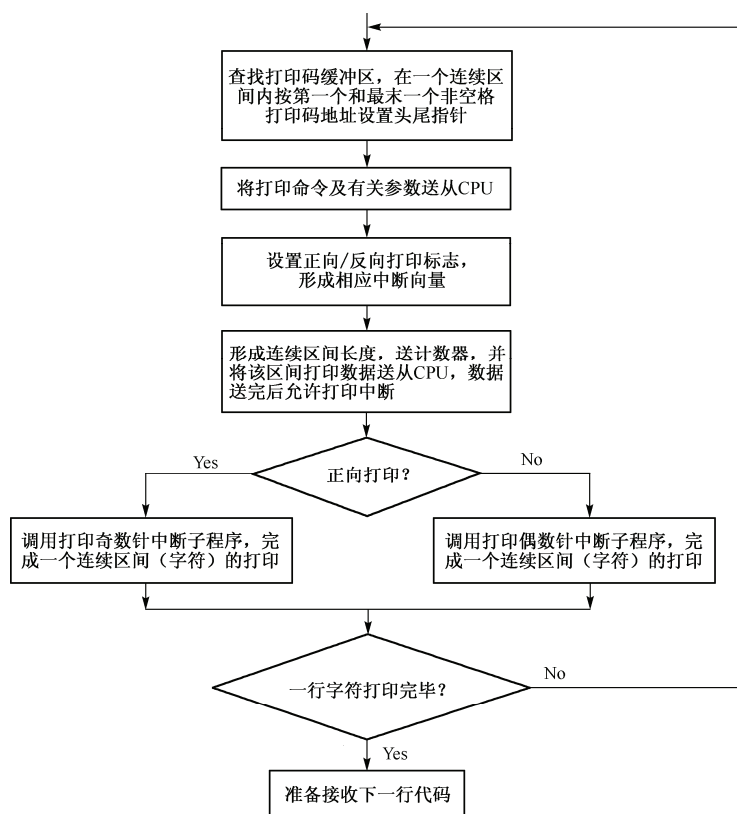


图 6-37 打印机的打印处理流程

连续区间，即第一个字符打印完后，再进行第二个连续区间的打印，直至一行数据打印完毕，准备接收下一行待打印的代码。

6.4.4 喷墨打印机

喷墨打印机是类似于用墨水写字一样的打印机，可直接将墨水喷射到普通纸上实现印刷，如喷射多种颜色墨水可实现彩色硬拷贝输出。

喷墨打印机的喷墨技术有连续式和随机式两种，目前市场上流行的各种型号打印机大多采用随机式喷墨技术。早年的喷墨打印机及当前输出大幅面打印机采用连续式喷墨技术。

1. 连续式喷墨打印机工作原理

图 6-28 是一种电荷控制式打印机的印刷原理和字符形成过程。主要由喷头、充电电极、偏转电极、墨水供应及过滤回收系统和相应控制电路组成。其工作原理如下。

喷墨头后部的压电陶瓷受振荡电脉冲激励，产生伸缩，使墨水断裂形成墨滴而喷射出来，只要电脉冲存在，墨滴就能连续喷射出来。墨滴是不带电的，在其前面设置充电电极，施加静电场给墨滴充电。所充电荷的多少由字符发生器控制，根据所印字符各点位置的不同而充以不同的电荷。充电电极所加电压越高，充电电荷越多，墨滴经偏转电极后偏移的距离也越大，最后墨滴落在印字纸上。图 6-38(a)中只有一对偏转电极，因此墨滴只能在垂直方向偏移。若垂直线段上某处不需喷点，则相应墨滴不充电，在偏转电场中不发生偏转而射入回收器中，横向偏转则靠喷头相对于记录纸作横向移动实现。图 6-38(b)为墨滴落在纸上的顺序（H 字符），字符由 7×5 点阵组成。

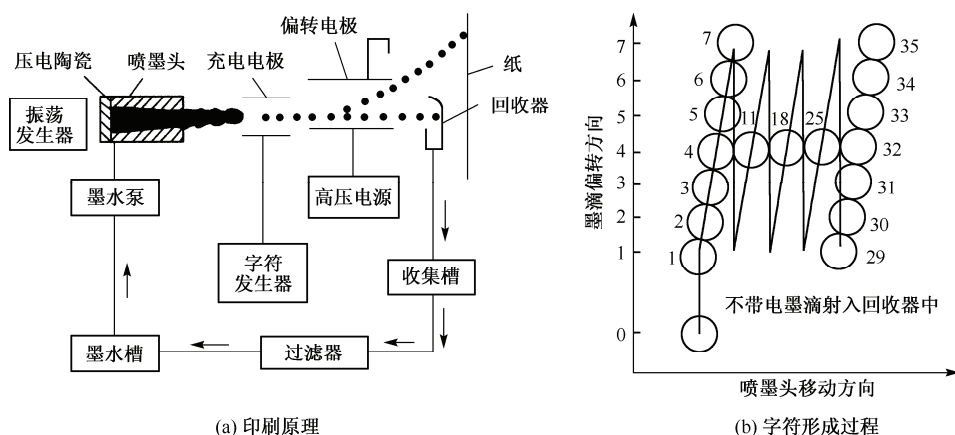


图 6-38 电荷控制式喷墨打印机

有的喷墨打印机采用两对互相垂直的偏转板，对墨滴的印字位置进行二维控制。上面介绍的打印机只有一个喷头，因此速度较慢。

2. 随机式喷墨打印机工作原理

这种系统供给的墨滴只在需要印字时才喷出，因此不需要墨水循环系统，省去了墨水泵和收集槽等。与连续式相比，随机式喷墨打印机结构简单、价廉，可靠性高。为提高印字速率，这种印字头采用单列、双列或多列小孔，一次扫描喷墨即可打印出所需的字符或图像。

产生墨滴的机构可采用不同的技术，流行的有压电式和热电式。

(1) 压电式喷墨技术

有压电管型、压电隔膜型和压电薄片型等多种。其机械结构虽不同，但有共同的特点，即加高电压脉冲于墨水压电换能器上，使墨水受力，生成墨滴并喷出，完成印字过程。喷出的墨滴是不连续的，墨滴的发生频率最大可超过 10 kHz。图 6-39(a)和(b)分别为压电管型喷墨结构和喷嘴结构示意图，现以它为例来说明喷墨技术的工作原理。

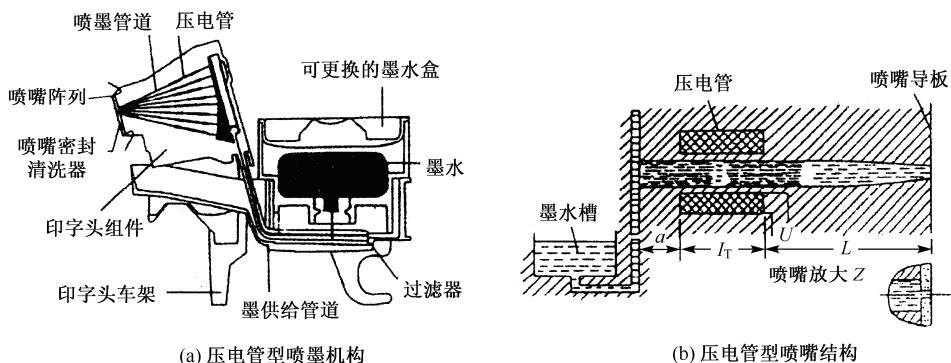


图 6-39 压电式喷墨打印机

压电式喷墨打印机由印字头组件、印字头车架和墨水盒等组成。印字头组件中可容纳几个到几十个喷嘴。其单个喷嘴的结构如图 6-39(b)所示，由压电管、墨水管道和喷嘴组成。墨水在容器中的水平位置低于喷嘴的最低位置，因毛细管作用使剩余的墨水保持在喷嘴内，不至于溢出到喷嘴外。压电管为换能器，60~200 V 脉冲电压作用其上，可使压电管内部截面积收缩或扩张，将墨滴喷出，多余的墨水被收回。生成墨滴的周期约 200 ns。

(2) 热电式喷墨技术

热电式也称为气泡式，其换能元件为加热器件，可采用电阻。图 6-40 展示了气泡式墨滴的基本生成过程。

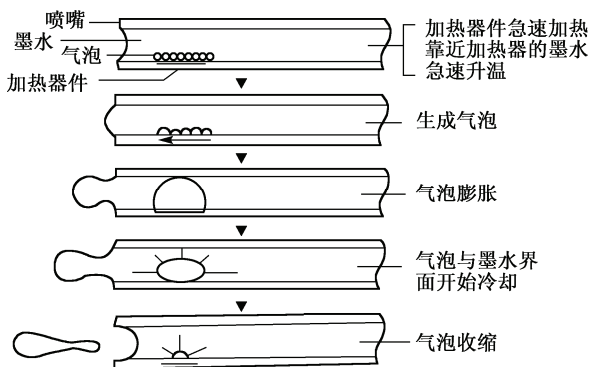


图 6-40 气泡式墨滴的生成过程

3. 喷墨打印机的消耗品及辅助装置

喷墨打印机所选用的墨水、记录纸（打印纸）以及印字头的清洗密封装置与印字质量有密切关系。喷墨打印机需要特殊的墨盒，其价格较贵。彩色喷墨打印机有 4 只分别装有品红、黄、青、黑色的墨水盒。

(1) 墨水

墨水有严格的性能要求，不能随意更换不同规格的墨水，否则会对喷墨系统造成致命的损害，甚至报废。随机式喷墨机构一旦发生故障，是不可修复的。

对墨水的要求如下：① 墨水所用原料，必须满足印字机构要求，不应含有颗粒等；② 墨水在管道和喷嘴内，不应有渗漏、沉积或干涸现象；③ 能在 $-25^{\circ}\text{C} \sim +70^{\circ}\text{C}$ 温度范围内存放；④ 在 $+10^{\circ}\text{C} \sim +40^{\circ}\text{C}$ 温度范围内，应符合规定的浓度、黏度和表面张力值。

印字质量对墨水的要求如下：印字清晰，不产生模糊边缘，快干，印字有防水、不褪色和不可擦除的性能。

(2) 记录纸（打印纸）

一般喷墨打印机要求使用与规定墨水要求相符的普通纸（复印纸），对穿透性和渗析性有一定要求。穿透性是指墨滴在纸面上向另一面的穿透能力，以不穿透到另一面为佳。渗析性也称为浸润性，会使印刷输出字符的边缘产生浸润模糊。另外，注意不要因纸的质量问题而引起细微喷嘴的堵塞。

(3) 辅助装置

喷墨打印机带有一些通用的或专用的选件和辅助装置，如接口卡、输纸机构、供纸和接纸盘等。为确保喷嘴不受阻塞，对使用水性墨水的喷墨打印机应带有清洗装置，定期（或遇有故障）对印字机构进行清洗。由于水性墨水容易干涸造成喷头阻塞，有些厂商开发了静电型使用油型墨水的印字头。靠静电的吸引力，使高温沸腾（ 200°C ）的油性墨水从喷嘴中喷出，这样可省去清洗装置。使用油性墨水的打印机可进一步提高分辨率。

6.4.5 激光打印机

随着电子摄影技术和激光扫描技术的发展，一种新型的非击打式打印输出设备——激光打印

机在 20 世纪 70 年代中期问世。这种新型的打印机的优点为：打印速度快，印字质量好，分辨率高，噪声小，能打印各种字符、图形、汉字、图表等。如美国 HP 公司的 HP Laserjet P3015 型（最高分辨率 1200×1200 dpi、最高打印速度 40 ppm、黑白激光打印机）、日本施乐公司的 DocuPrint211 型（最高分辨率 1200×1200 dpi、最高打印速度 22 ppm 类型的、黑白激光打印机），比击打式打印机提高性能几十倍至几百倍。特别是在半导体激光器和高灵敏度感光材料研制成功后，激光打印机的体积大大缩小，各种小型台式、多功能、低价格的中、低速激光打印机相继出现，并能够与大、中、小、微型计算机配套，成为一种理想的办公自动化设备和汉字输出设备。

激光打印机主要由激光扫描系统、电子摄影系统和控制系统组成，如图 6-41 所示。激光打印机利用激光扫描技术将经过调制的、载有字符点阵信息或图形信息的激光束扫描在光导材料上，并利用电子摄影技术让激光照射过的部分曝光，形成图形的静电潜像。再经过墨粉显影、电场转印和热压定影，便在纸上印刷出可见的字符或图形。

1. 主要组成部分

(1) 激光扫描系统

激光扫描系统由激光器、调制部件、扫描部件和光学部件组成。它的功能是对激光器产生的激光按字符点阵信息或图像信息进行高频调制，将电信号转变成光信息。调制后的载有信息的激光束经过光学部件整形、聚焦，由扫描部件控制在光导材料上逐行扫描，从而形成静电潜像。

激光器是激光打印机的光源，在受激时能发射出一种具有单色性和单方向性的强辐射光——激光。激光可以聚焦成极细的光束，特别适于记录高分辨率的信息。早期的激光打印机多采用氦氖气体激光器，由于体积大、成本高，使激光打印机的应用范围受到一定限制。20 世纪 70 年代末，随着激光二极管的出现，这种体积小能直接进行高频调制的半导体激光器便广泛地被用于各种小型激光打印机中。

对氦氖激光器需进行偏转调制，可采用机械、电光或声光的方法实现。对半导体激光器可直接按照字符或图像的二进制信息进行开关调制，将字符点阵代码通过调制器转变为激光驱动信号，控制激光二极管接通或断开。

激光扫描部件一般采用转镜扫描器，这是一种多面棱镜，由扫描马达带动匀速旋转。当激光束照射在转镜的某一面时，随着镜子的转动，反射光便扫描成一条直线，如图 6-42 所示。当转镜旋转到下一面时，反射光随着镜子的转动又扫描成一条直线。因此，若转镜有 n 面（如 6 面），旋转一周，便扫描 n 条（6 条）直线。

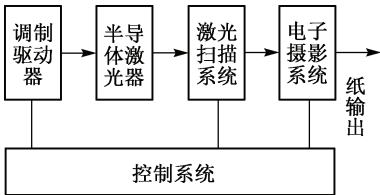


图 6-41 激光打印机组成框图

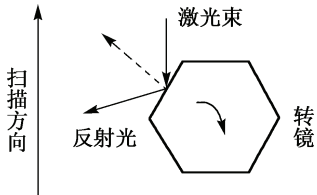


图 6-42 转镜扫描原理

(2) 电子摄影系统

电子摄影系统包括感光鼓充电器、显影磁刷、转印分离部件、定影器及消电清洁装置等。

感光鼓是一个用铝合金制成的圆筒，表面涂有一层光导材料（如硒）。这种光导材料在黑暗中为绝缘体，若被光照射，其电阻值下降，成为导体。感光鼓预先在暗处充电，以便当载有字符信息的激光束扫描鼓面时，其光照部分的电阻值降低，使电荷流失，而未照部分保持电荷，这样

便在鼓面形成字符的静电潜像。

显影磁刷中装有带电荷（与静电潜像极性相反）的墨粉，当磁刷接触鼓面时，墨粉被潜像吸附，形成可见图像。

充电器、转印器、分离器和消电器一般均采用电晕丝，通过电晕放电实现对感光鼓充电、将墨粉图像转印在纸上、将纸与感光鼓分离以及印刷完成后消除感光鼓表面残留的电荷等。

墨粉转印到纸上后容易被擦掉，需要经过定影将墨粉图像固定在纸上。可用热辊作定影器。当纸接触到热辊时，墨粉受高温立即溶化而黏附在纸上，并经加压形成固定图像。

（3）控制系统

控制系统对激光打印机的整个打印过程进行控制，包括控制激光器的调制；控制扫描电动机驱动多面棱镜匀速转动；进行行同步信号检测，控制行扫描定位精度；控制步进电动机驱动感光鼓等速旋转，保证垂直扫描精度，使激光束每扫描一行都与前一行保持相等的间距。此外，还对显影、转印、定影、消电、走纸等操作进行控制。接口控制部分接收和处理主机发送来的各种信号，并向主机回送激光打印机的状态信号。

2. 激光打印过程

一台激光打印机的完整打印过程如图 6-43 所示。随着感光鼓的旋转，一次完整的打印过程一般分成 7 步进行。

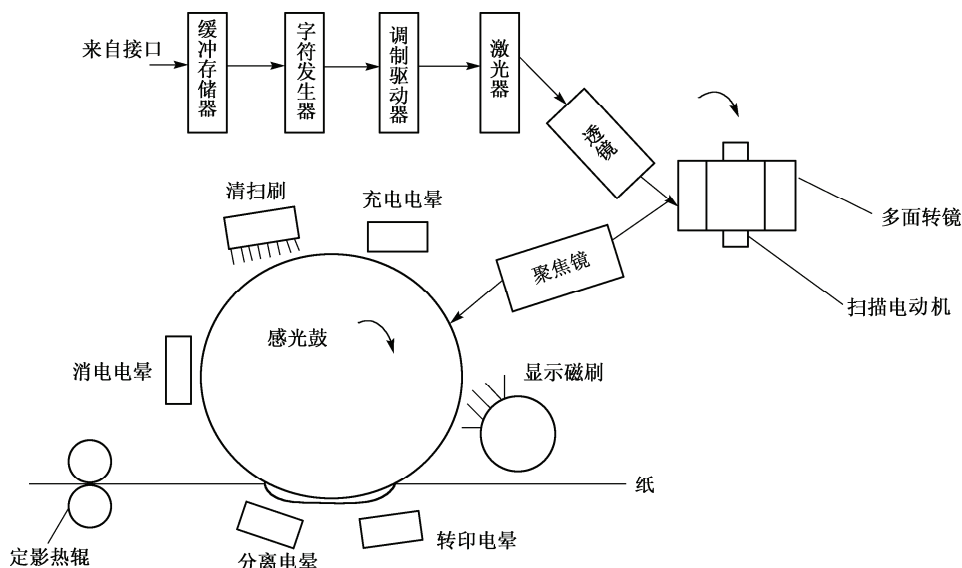


图 6-43 激光打印机的打印过程

（1）充电

预先在暗处由充电电晕靠近感光鼓放电，使鼓面充以均匀电荷。

（2）曝光

主机输出的字符代码经接口送入激光打印机的缓冲存储器，通过字符发生器转换为字符点阵信息。调制驱动器在同步信号控制下，用字符点阵信息调制半导体激光器，使激光器发出载有字符信息的激光束。这种激光束是发散的，经透镜整形成为准直光束，并照射在多面转镜上，通过聚焦镜将反射光束聚焦成所需要的光点尺寸，然后光束沿感光鼓轴线方向匀速扫描成一条直线。

当充有电荷的鼓面转到激光束照射处时，便进行曝光。由于激光束已按字符点阵信息调制，

使鼓面上不显示字符的部分被光照射。光照部位电阻下降,电荷消失,其他部位仍然保持静电荷,于是,在鼓面形成一行静电潜像。转镜每转过一面,由同步信号控制重新调制激光束,并在旋转的鼓面上再次扫描,形成下一行静电潜像。

(3) 显影

当载有静电潜像的感光鼓面转到显影处时。磁刷中带相反电荷的墨粉便按鼓面上静电分布的情况,被吸附在鼓面带有电荷的部位,从而在鼓面显影成可见的字符墨粉图像。

(4) 转印

墨粉图像随鼓面转到转印处,在纸的背面用转印电晕放电,使纸面带上与墨粉极性相反的静电荷,于是墨粉靠静电吸引而黏附到纸上,完成图像的转印。

(5) 分离

在转印过程中,静电引力使纸紧贴鼓面。当感光鼓转至分离电晕处时,用电晕不断地向纸施放正、负电荷,消除纸与鼓面因正、负电荷所产生的相互吸引力,使纸离开鼓面。

(6) 定影

将分离后的纸送定影热辊。墨粉中含有树脂,纸接触到热辊时树脂被加热而溶化,使墨粉紧贴在纸上,同时使热辊挤压纸面。这样,经过热压处理,图像便牢牢地定影在纸上。

(7) 消电和清洁

完成转印后,感光鼓表面还留有残余的电荷和墨粉。当鼓面转到消电电晕处时,利用电晕向鼓面施放相反极性的电荷,使鼓面残留的电荷被中和掉。感光鼓再转到清扫刷处,刷去鼓面的残余墨粉。这样感光鼓便恢复原来的状态,又可开始一次新的打印过程。

6.4.6 打印机适配器

与显示器的适配器不同,打印机的适配器并不包含打印机控制器。打印机的适配器接口标准常见的一般有并行接口和串行接口这两种。

并行接口一般称为 Centronics 接口,也称为 IEEE 1284,能通过标准的并口实现主机与打印机之间的信息交换。打印机的并行接口适配器的组成和工作原理已在第 5.4.7 节中作为例子介绍过了,这里不再赘述。

在 IEEE1284 标准中定义了多种并行接口模式,常用的有以下三种:① SPP (Standard Parallel Port),标准并行接口;② EPP (Enhanced Parallel Port),增强并行接口;③ ECP (Extended Capabilities Port),扩展功能并行接口。这三种模式的硬件和编程方式不同,传输速率为 50KB/s~2MB/s。一般情况下,打印机一侧采用 Centronics 36 针弹簧式接口,主机一侧常采用 25 针 D 型接口。因这种 25 针并行接口尺寸小,也有一部分小型的打印机(如 POS 机的打印机)采用。

另外一种打印机适配器接口就是串行接口,其特点是传输速率慢但传输距离长。现在的台式计算机一般有两个串行口 COM1 和 COM2,通常 COM1 是 9 针 D 形接口(即 RS-232),而 COM2 使 25 针接口(即 RS-422)。此外,打印机一般都还支持另外一种特殊的串口方式,即 USB 接口(Universal Serial Port)。普通 USB 接口能提供 12 Mbps 的连接速度,比早期的并口速度提高 10 倍以上,能使打印文件的传输时间大大缩减。如果采用 USB 2.0 标准,还能进一步将接口速度提高到 480 Mbps,能进一步大幅减少打印文件的传输时间。

目前的打印机中,针式打印机一般都能支持 IEEE 1284 并口和 USB 接口等,主流的喷墨打印机和激光打印机一般都只支持 USB 接口。

在世界范围内,打印机由击打式向非击打式发展已成为定局。击打式打印机的最大优势是整

机价格较低,尤其是消耗品(如色带)费用低。但由于非击打式打印机已大幅度降价,使击打式打印机已基本退出了市场。非击打式打印机的打印质量(尤其是彩色打印机)好,噪声低,但消耗品(如激光打印机和喷墨打印机的墨盒)费用高,也使部分用户难以承受。

虽然目前激光打印机是页式印字机中技术最成熟的机种,但并不是最佳的机种。激光打印机如要进一步提高印字质量,受到激光产生部件中的多面转镜运动速度及其可动部分精度的限制。从技术发展观点出发,今后激光打印机将面临其他机种的激烈竞争。

此外,针式打印机随着市场需求的变化也在逐步调整其产品方向。目前除了在大型宽行报表打印方面,针式打印机的定位还逐渐转向了微型打印和特种打印。例如,打印超市的购物小票、银行的存款单、酒店账单、航空公司的机票、铁路的车票和货运单以及海关的报关单据等。因为中低档激光打印机和喷墨打印机仅可打印 A3 和 A4 幅面以下的纸张,且对纸张的厚度还有要求,也不能实现多联打印,而针式打印机在这些方面都具有不可被取代的优势。目前,新型针式打印机的打印分辨率最高可达到 360 dpi。

随着 Windows 操作系统在计算机领域的普及,图形界面已成为计算机系统及应用软件的人机界面。在这种界面下,文字和图形都采用图形打印方式,因此打印机的另一重要发展趋势是从文本打印方式向图形打印方式转变,从单色打印向彩色打印转变。

6.4.7 3D 打印技术简介

3D 打印是一种快速成型的打印技术,以数字模型文件为基础,运用粉末状金属或塑料等可黏合材料,通过逐层打印的方式来构造物体。原理上 3D 打印与普通打印基本相同,打印机内装有黏合液体或粉状金属、粉状塑料、陶瓷颗粒等可黏合的“打印材料”,通过计算机控制,把“打印材料”一层层叠加起来,最终把数字模型变成实物。

1. 发展历程

3D 打印技术出现在 20 世纪 90 年代中期,实际上是利用光固化和纸层叠等技术的最新快速成型装置。它与普通打印工作原理基本相同,打印机内装有液体或粉末等“打印材料”,与电脑连接后,通过电脑控制把“打印材料”一层层叠加起来,最终把计算机上的蓝图变成实物。这打印技术称为 3D 立体打印技术。

1986 年,Charles Hull 开发了第一台商业 3D 印刷机。

1993 年,麻省理工学院获 3D 印刷技术专利。

1995 年,美国 ZCorp 公司从麻省理工学院获得唯一授权并开始开发 3D 打印机。

2005 年,市场上首个高清晰彩色 3D 打印机 Spectrum Z510 由 ZCorp 公司研制成功。

2010 年,世界上第一辆由 3D 打印机打印而成的汽车 Urbee 问世。

2011 年,发布了全球第一款 3D 打印的比基尼。

2011 年,英国研究人员开发出世界上第一台 3D 巧克力打印机。

2011 年,南安普敦大学的工程师们开发出世界上第一架 3D 打印的飞机。

2012 年,苏格兰科学家利用人体细胞首次用 3D 打印机打印出人造肝脏组织。

2013 年,全球首次成功拍卖一款名为“ONO 之神”的 3D 打印艺术品;美国德克萨斯州奥斯汀的 3D 打印公司“固体概念”(Solid Concepts)设计制造出了 3D 打印的金属手枪。

2. 打印原理

3D 打印技术也被称为“增材制造”或“累积制造”。“增材制造”的理念不同于传统的“去

除型”制造。传统“去除型”数控制造一般是在原材料基础上，使用切割、磨削、腐蚀、熔融等办法，去除多余部分，得到零部件，再用拼装、焊接等方法组合成最终产品。3D 打印的“增材制造”与之截然不同，不需原胚和模具，就能直接根据数字模型数据，通过叠加材料的方法直接制造与数字模型完全一致的三维物理实体模型。

3D 打印实物的主要流程是：首先运用计算机软件进行三维建模，再将建成的三维模型“分区”为逐层的截面数据（即切片数据），模型切片以特定的文件格式保存在计算机中，以便指导 3D 打印机进行逐层打印。3D 打印机与计算机连接后，3D 打印机在计算机的控制下，根据模型文件的截面数据，用液体状、粉状或片状的材料将这些截面逐层地打印出来，再将各层截面以各种方式粘合起来从而“打印”出模型实体。

3D 打印机打出的截面的厚度（即 Z 方向）以及平面方向即 X-Y 方向的分辨率是以 dpi 或者 μm 来计算的，一般的厚度为 $100\ \mu\text{m}$ 。有时打印出来的实物弯曲表面比较粗糙（像计算机图像上的锯齿失真一样），如要获得更高分辨率的物品，可以先用 3D 打印机打印出稍大一点的物体，再经过表面轻微打磨即可得到表面光滑的“高分辨率”实物。

3. 发展趋势

3D 打印技术的特点在与使用该技术几乎可以造出任何形状的物品。

过去这种技术经常在模具制造、工业设计等领域被用于制造模型，然后再根据模型制模来进行大批量产品或零件生产。现在 3D 打印技术逐渐开始用于一些产品的直接制造，已经有使用这种技术直接打印而成的零部件。采用该技术可以简化产品的制造程序，缩短产品的研制周期，提高效率。

目前，3D 打印技术主要的打印材料为塑料，金属等其他打印材料由于价格昂贵和工艺复杂，使用得也比较少。因为受限于可使用的打印材料的种类，所以 3D 打印技术还无法应用于大批量产品的工业化生产。由于打印机本身及打印材料费用都非常高，故 3D 打印机的普及化、家庭化在短期内也较难实现。如果未来突破这些限制，3D 打印技术的发展和普及将更迅速。

6.5 其他输入/输出设备

20 世纪 70~80 年代微型计算机与人进行信息交流的能力不强，只能通过键盘输入信息，在屏幕上显示结果或者在打印机上输出。人类在进行信息交流时习惯于用眼睛看，用耳朵听，通过语言文字进行信息交流。随着科学技术的发展，具有在更高级视觉听觉能力上与人交流的计算机系统——多媒体计算机系统开始出现，能直接输入文字、图形图像，能听懂人们的讲话，同时能直接用语音合成系统发出声音，能播放高质量的音乐，直接用影视动画向人们表达各种信息。

6.5.1 光学字符识别设备

自 20 世纪中期开始进行数字、符号识别技术的研究以来，文字识别目前已达到实用化阶段。以美国、日本为代表的 OCR（Optical Character Recognition，光学字符识别）技术反映了当今世界的先进水平。

在美国，由于普遍使用打字机，因此识别的对象多是印刷体英文、数字和某些符号。大部分扫描仪配备了英文 OCR 软件，用户可方便地将英文文本送入计算机。日本文字有表音文字（假名）和表意文字（汉字），需要识别的对象有印刷体日本文字和手写体日本汉字，其文字识别技

术的困难和复杂程度是拉丁文字所无法比拟的。我国的汉字常用的也有几千个，这也增加了文字自动识别的困难。

OCR 系统是多项技术结合的产物，识别技术是其核心内容，还包括：图形文本的扫描输入，光电信号变换，电信号的数字化处理，版面分析与理解，字的切分处理，以及输入信息载体（页）的自动传输技术等。

字的切分直接影响字的正确识别率。字的切分是从文本图像中将每个字符正确切分下来，汉字切分的依据是每个汉字是大小均匀的方块字，但在汉字文本中经常混有大小写英文字母和标点符号，这些字符的高低位置与字宽都与汉字不一样；同时汉字中包括了许多左右可分的汉字，有可能将其切分成两个字，如将“张”切分成“弓”和“长”。另外，由于印刷质量和噪声干扰，产生笔画的模糊和断裂，也可能造成切分的错误。

我国文字识别技术的研究始于 20 世纪 70 年代末，相继推出了自动阅卷机等英文、数字实用 OCR 系统和几个实用的印刷体汉字识别系统，并正在进行手写体汉字识别的理论和实用化研究工作，如今国内做得较好的识别软件有北京汉王科技公司的尚书汉字表格识别系统、北京清华紫光文通信息技术有限公司的 TH-OCR 文通手写识别系统等。

使用文字识别技术的扫描仪是一种光机电一体化的高科技产品，是将各种形式的图像信息输入计算机的重要工具，是继键盘和鼠标之后的第三代计算机输入设备，也是功能极强的输入设备。从最直接的图片、照片、胶片到各类图纸图形以及各类文稿资料都可以用扫描仪输入到计算机中，进而实现对这些图像形式信息的处理、管理、使用、存储、输出等。配合文字识别软件，还可以将扫描的文稿转换成计算机的文本形式。目前，扫描仪已广泛应用于各类图形图像处理、出版、印刷、广告制作、办公自动化、多媒体、图文数据库、图文通信、工程图纸输入等领域，极大地促进了这些领域的技术进步，甚至使一些领域的工作方式发生了革命性的变革。

自然界每种物体都会吸收特定的光波，没有被吸收的光波就会被反射出去。扫描仪就是利用这种特性来完成对稿件的读取的。扫描仪在工作时会发出强光照射在稿件上，没有被吸收的光线将被反射到光学感应器上。光学感应器接收到这些信号后，再将这些信号传输到模数转换器，模数转换器再将其转换成计算机能够读取的信号，然后通过驱动程序转换成显示器上能看到的正确图像。欲扫描的稿件通常可以分为反射稿和透射稿。反射稿泛指一般的不透明文件，如报纸、杂志等。透射稿包括幻灯片（正片）或底片（负片）。如果经常需要扫描透射稿，就必须选择具备光罩（光板）功能的扫描仪。

扫描仪的光学读取装置相当于人的眼球，其重要性不言而喻。目前，扫描仪所使用的光学读取装置有两种：CCD 和 CIS。

6.5.2 图形图像输入设备

常见的用于图形图像输入的设备有光笔、画形板、画笔、数字相机等。

1. 光笔

光笔的外形如笔状，用来检测光信号，因此而得名。光笔由透镜组、光导纤维、触钮开关等组成，其结构如图 6-44 所示。光笔可定位和跟踪，与键盘配合使用，可以画出图形，进行编辑和修改，是计算机辅助设计的有力工具。

光笔的基本功能有拾取和跟踪两种。拾取是指在显示屏上有图形的条件下，用光笔选取一图形元素为参考点，对图形实施处理的过程。跟踪是指在显示屏上出现一个光标时，用光笔移动光

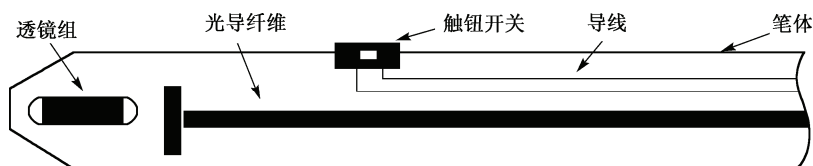


图 6-44 光笔的结构

标、实施定位的过程。脱机时，显示器上的光笔系统产生十字光标，并可用光笔拖动光标满屏移动，同时提供所指点的字符或图形曲线的坐标位置及数据，可将这些参数发送到主机处理。

2. 图形板和画笔

画笔呈笔状但不是光笔，用于图形板。当画笔接触到图形板的某一位置时，图形板把画笔的位置坐标自动变为数字信号发送给主机。图形板是一种二维的 A/D 变换器，图形板和画笔并用的方法称为数字化仪。画笔位置的测量方法有电阻式、电容式和电磁感应式等。在利用画笔和图形板进行图形处理的系统中，实际上常常要求把图形板上的画笔位置在显示屏上由光标显示出来。

为了提高读图精度，常用流动标代替画笔与图形板配合使用。流动标是一种手持的方形坐标读入器，在流动标上有一块透明玻璃，玻璃上刻有十字标记，十字标记的中心即是流动标的中心，将十字中心对准图形的坐标点，就能精确地读取坐标。

图形板与画笔配合的输入方式，比光笔与屏幕结合的输入方式有更多优点：光笔不能直接输入纸上的图形信息，图形板则很容易做到；光笔和持笔的手会挡住图形，又因屏幕玻璃厚度产生光的折射和人眼与光笔的视角误差等影响，常使画出的图形偏离预计的位置，图形板却无此问题。

图形板和画笔主要用于输入工程图。将图纸贴在图形板上，画笔沿图纸上的图形移动，即可输入工程图。

3. 数字照相机

近年来推出的数字照相机在色彩质量和图像清晰度等方面不断改进，但还不能与基于胶片的相机所能提供的最佳清晰度和色彩相媲美。

数字照相机内置几百 MB~几 GB 的存储器，并带有 LCD 预映屏幕。在拍照时观察屏幕可将最佳快照录入相机的存储器中，并即时删除不理想的照片。其最大的特点是可以通过 USB 接口联机，如果照片拍得不理想，则可把它们上传到计算机中，对其进行修改编辑，直到满意为止。因此，人们可以在瞬间获取图像，也可在照片拍摄后把其上传到因特网上。

数字照相机可以减少图片成像成本，迅速、便捷以及在网上发送新采集的图像。人们再也不必拍上一卷照片，以求获得一到两张有保存价值的照片。

4. 视频录像机

视频录像机是一种视频输入设备，在过去被广泛用于视频会议、远程医疗及实时监控等方面。近年来，由于感光成像器件（光学传感器）技术的成熟和生产规模的扩大，摄像头的造价大幅降低，目前，对于笔记本电脑、平板电脑、智能手机，摄像头已经成为标准配置。随着互联网网络速度的不断提高及互联网视频软件的发展，人们可以通过摄像头、话筒在网络上进行有影像、有声音的交谈和沟通。

视频录像机的主要部件有镜头、图像传感器、数字信号处理芯片和电源。

（1）镜头（Lens）

镜头是一种透光器件，与传统相机及数码相机的镜头原理一样，由几片透镜组成，有塑胶透

镜或玻璃透镜。

(2) 图像传感器 (Sensor)

感光器件将镜头透射的光学信号转换为电信号，并通过模数 (A/D) 转换为数字图像信号，有 CCD (Charge Couple Device, 电荷耦合器件) 和 CMOS (Complementary Metal Oxide Semiconductor, 互补金属氧化物半导体) 两大类。

(3) 数字信号处理芯片 (DSP)

数据处理部件对图像传感器转化的数字图像信号进行加工处理。

(4) 电源

摄像头内部需要两种工作电压：3.3 V 和 2.5 V，最新工艺芯片只需 1.8 V。

视频录像机工作的时候，景物通过镜头透射到图像传感器表面上。传感器先将接收到的光学视频转换电信号，再经过模-数 (A/D) 转换转换后变为数字视频信号，最后，传感器将数字视频信号传输到数字信号处理芯片。DSP 芯片将处理后的视频数据通过 USB 接口传输到计算机中，经过计算机的处理就能产生可视的视频影像。

6.5.3 语音识别设备

语音不仅是人类交流信息最自然、最有效的手段，也是人机通信的最有效的一种方法，摆脱了各种传统输入方法的弱点，比先进的手写体联机识别方式更优越。

语音识别和文字识别都属于模式识别的范畴，有共同的基本原理，但输入技术和工具则大不相同。语音输入方式是利用声音的物理模型，通过语音分析手段，预先将一些语音的特征参数提取，并存于计算机系统中。当讲话者语音输入时，处理系统就对该信号进行特征参数抽取，然后与已存储的特征参数进行比较，通过逻辑判别或距离测量，对输入的语音做出识别。语音识别流程如图 6-45 所示。

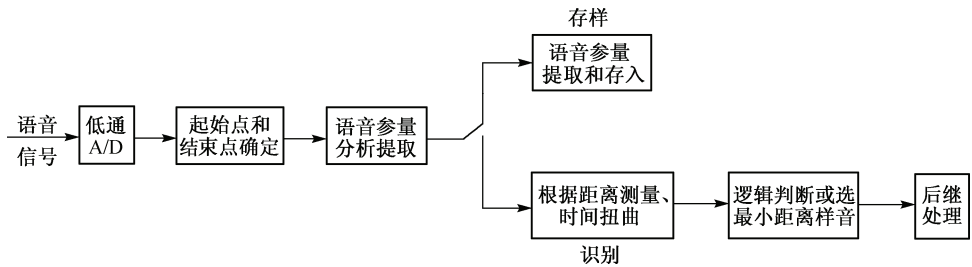


图 6-45 语音识别流程

通过话筒或录音输入设备输入的语音信号是模拟量，经放大、滤波和进行 A/D 转换后，得到这个语音信息的数字编码，才能送入计算机进行处理。精确地测定一个语音的起始点和结束点，是语音参量与样音参量比较的基础。为了便于测量，常按每 10 ms 为一帧的时间顺序来进行，并由粗判和细判两个步骤来具体确定。首先进行粗判：由某帧开始，若连续 NF 帧的参数满足粗判有声条件，则判为头部有声；此后，若连续 NR 帧满足粗判无声条件，则判为尾部无声。然后进行细判：从粗判起始点前 25 帧开始向后搜索，当从某帧开始连续 SNF 帧满足细判有声条件，该帧确定为起始点；尾部判定从粗判无声后的 15 帧开始向前搜索，连续 SNR 帧为无声，该帧被判定为结束点。

当进行语音识别时，时间的扭曲处理是极为重要的一步。准确地测定语音的起始点和终点，并从中提取了该区间有关语音参量后，还不能立即与样音进行比较。这是因为，即使是同一个人

重复发同一个音，每次发音也不可能完全一致，即各帧参量不可能完全相同，为了消除各帧由离散性带来的影响，使被测语音尽可能地与样音对齐，就必须将它们中的一个进行时间扭曲处理。经过扭曲处理后，被测语音强度曲线和样音强度曲线的吻合程度得到改进。这时就可以转入识别处理，即选出各参量最为接近的样音的语言信号作为最后输出。

此外，还有一种识别方法是按照一定规则把测得的语音特征进行计算，译成单词、短语和句子。这种方法较为复杂，但前途极广。

6.5.4 条形码与二维码识别仪

1. 一维条形码

一维条形码又叫条码。条码的定义是：由一组宽度和反射率不同的平行相邻的“条”和“空”，按照预先规定的编码规则组合起来，用来表示一组数据的符号。这组数据可以是数字、字母或某些符号。图 6-46(a)为条码符号基本术语示意图，解释如下：

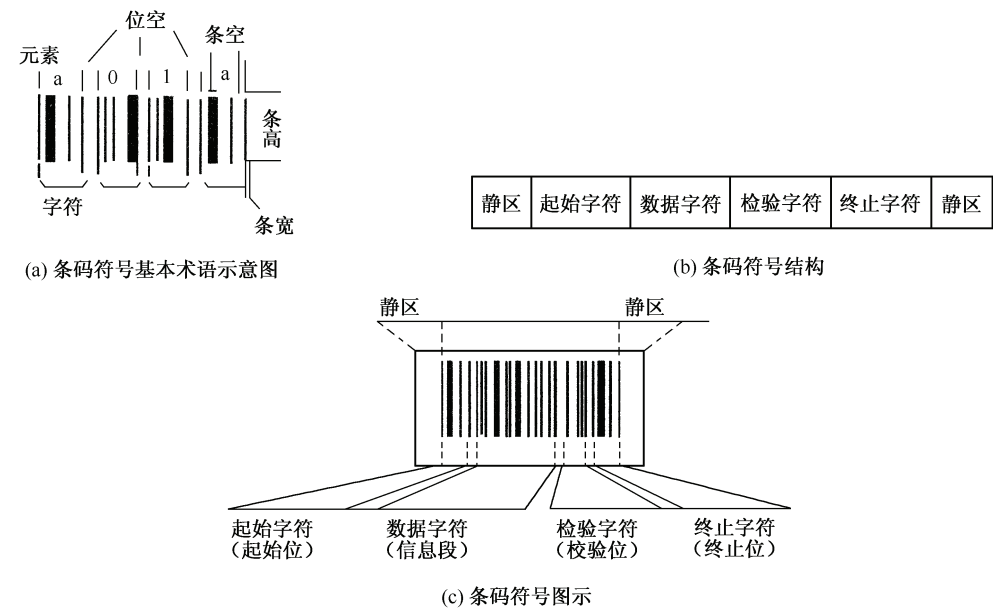


图 6-46 条形码的基本结构

条码元素：用来组成条码符号的条和空，统称条码元素。

条：在条码符号中，反射率较低的元素称为条，即图中的黑条。

空：又称为间隔。在条码符号中，反射率较高的元素称为空，即图中的白条。

条与空可规定几种不同的宽度，由若干条和空组成一个字符。

条码的作用：条码是表示数据的符号，在市场上即表示某种商品的代号，这个符号可由机器自动识别，并送入计算机中。超市中的商品都用条码识别。条码也可应用于仓库管理、图书管理等领域。

条码技术是研究如何把计算机所需要的数据用条码来表示，并将它转变为计算机可自动采集的数据。因此，条码技术主要包括：条码编码规则及标准、条码译码技术、印刷技术、光电扫描技术、通信技术、计算机技术等。

需要一个专用的扫描装置来要阅读条码包含的信息。目前，条码中最窄元素的宽度可小到

0.15 mm, 扫描器的分辨率应与之对应。正常工作时, 扫描器与条码符号之间要保持的距离称为工作距离, 而最大工作距离和最小工作距离之差则称为扫描景深。

扫描器将扫描得到的脉冲数字信号送到译码器, 译码器按照一定的编码规则解释成计算机可识别的信号, 并通过数据通信接口(常用串行口 RS-232/485/422)送入计算机。通常, 条码译码器采用微处理器及相应硬件来完成译码工作。在 20 世纪 80 年代后期就已有专用译码芯片上市。

扫描器和译码器合起来称为阅读器。

条形码符号的结构如图 6-46(b)和(c)所示。

一个完整的条码符号由两侧静区、起始字符、数据字符、校验字符(可选)和终止字符组成。在印制条码符号时要留出静区, 在扫描条码时要从静区开始, 以保证阅读效果。条码符号的第一个字符是起始字符, 是特殊规定的条空结构, 在它后面才是真正的数据字符。条码符号的最后一个字符是终止字符。如选用的条码还要求有校验字符, 则校验字符位于数据字符与终止字符之间。有的条码符号在字符之间留有“位空”, 如图 6-46(a)所示, 有的则不留“位空”, 分别称为离散型条码和连续型条码。

2. 二维方形码

二维方形码是指在一维条码的基础上扩展出另一维具有可读性的条码, 使用黑白矩形图案表示二进制数据, 被设备扫描后可获取其中所包含的信息。

一维条码的宽度记载着数据, 而其长度没有记载数据。二维条码的长度、宽度均记载着数据。二维条码有一维条码没有的“定位标志”和“容错机制”。容错机制在即使没有辨识到全部的条码、或是说条码有污损时, 也可以正确地还原条码上的数据信息。

二维条码的种类很多, 不同的机构开发出的二维条码具有不同的结构以及编写、读取方法。按照编制原理, 二维码可以分为行排式二维码和矩阵式二维码两大类。

(1) 行排式二维码

行排式二维码(又称为堆积式二维码或层排式二维码), 其编码原理是建立在一维条码基础之上, 按需要堆积成二行或多行。行排式二维码在编码设计、校验原理、识读方式等方面继承了一维码的一些特点, 识读设备与条码印刷与一维条码技术兼容。但由于行数的增加, 需要对行进行判定、其译码算法与软件也不完全相同于一维码, 具有代表性的行排式二维码有 CODE 49、CODE 16K 和 PDF 417 等。

(2) 矩阵式二维码

矩阵式二维码又称为棋盘式二维码, 是在一个矩形空间中, 通过过黑、白像素在矩阵中的不同分布位置来进行编码的。在一个矩阵元素位置上, 通过出现方点、圆点或其他形状的点来表示二进制“1”, 不出现这些点则表示二进制的“0”, 因此可以通过点的排列组合来确定矩阵式二维码所代表的数据内容及其含义。

矩阵式二维码是建立在计算机图像处理技术和组合编码原理等基础上的一种新型图形符号自动识读处理的一种码制。具有代表性的矩阵式二维码有 Code One、Maxi Code、Data Matrix 和 QR (Quick Response) 码等, 其中以 QR 码最常用。

QR 码于 1994 年由日本 Denso Wave 公司发明。QR 是 Quick Response 的缩写, 即快速响应, 希望 QR 码的内容可以快速被解码。QR 码最先流行于日本, 目前也是我国最流行的二维码。QR 码可以比一维条码存储更多数据, 也不需像一维条码般在扫描时需要垂直直线对准扫描仪。

QR 码常见的是黑白两色, 呈正方形, 在正方形 3 个角落, 印有较小的像“回”字的正方图

案，这 3 个“回字”是帮助解码软件进行二维码位置探测的标志，用户不需要对准，无论以任何角度扫描，数据仍然可以被正确读取。一个完整的 QR 码包括二维码矩形图四周的静区、版本信息、格式信息、编码区和功能标志等。

QR 码有 40 个版本，版本 1 为 21×21 模块矩阵格式，版本 2 为 25×25 模块矩阵格式，以此类推，每个版本的模块数比前一版本的二维码图形矩阵的矩形边每边增加 4 个模块，直到版本 40，版本 40 的规格为 177×177 模块矩阵。图 6-47(a)为版本 7 (45×45 模块) 的 QR 码基本结构示意图，图 6-47(b)为一个阵列为 45×45 模块的 QR 码实例。

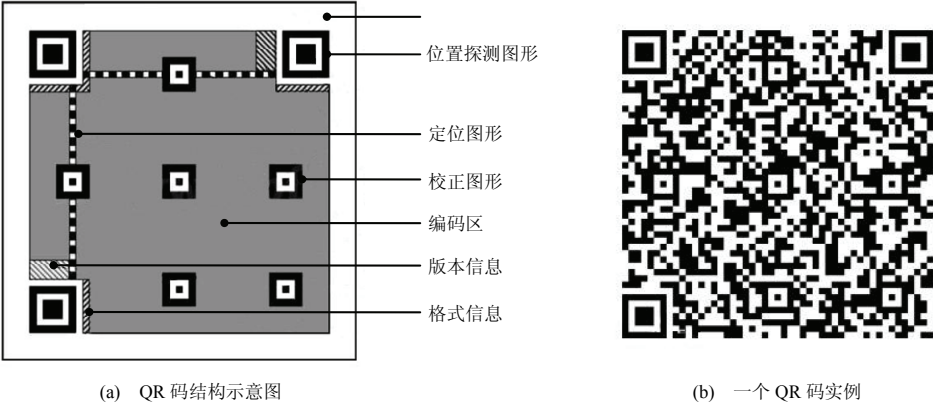


图 6-47 QR 码的基本结构及实例

使用配置有 QR 码解码软件的智能手机的摄像头，对着 QR 码扫描，解码软件就会自动解读 QR 码的图形信息。目前国内几乎所有带有摄像头的数码产品都具备 QR 码扫描及别功能。表 6-9 归纳了 QR 的规格、表达容量和纠错能力等一些基本特性。

表 6-9 QR 码的基本特性归纳

| | |
|--------------------------|---|
| 符号规格 | 21×21 模块（版本 1）～177×177 模块（版本 40），版本增加 1，每边增加 4 个模块 |
| 数据类型与容量(最大规格符号版本 40-L 级) | 数字数据，7089 个；字母数据，4296 个；8 位字节数据，2953 个；中国汉字、日本汉字，1817 个 |
| 数据表示方法 | 黑色模块，1；白色模块，0 |
| 纠错能力 | L 级，约可纠错 7%的数据码字；M 级，约可纠错 15%的数据码字 Q 级，约可纠错 25%的数据码字；H 级，约可纠错 30%的数据码字 |
| 结构链接（可选） | 可用 1～16 个 QR Code 码符号表示一组信息 |
| 掩模（固有） | 使符号中深色与浅色模块的比例接近 1:1，使因相邻模块的排列造成译码困难的可能性降为最小 |
| 扩充解释（可选） | 表示默认字符集以外的数据（如阿拉伯字符和希腊字母等），以及其他解释（如压缩方式）或者对行业特点进行编码 |
| 独立定位功能 | 有 |

目前，QR 码在国内外应用越来越广，如自动化文字传输、数字内容下载、网址快速链接、身份鉴别与商务交易等。2009 年底，中国铁路火车票就开始采用 QR 码作为防伪措施，取代了曾经的一维条码。四川成都、浙江杭州及河北石家庄等城市已陆续在公交站台上印刷 QR 码，提供乘客用智能手机扫描并识别线路和站点信息。我国目前大力推广的电子票务也是基于 QR 码进行的。通过短信方式发送一张包含相关信息的 QR 码发到用户手机，使用时用户只需在指定地点的二维码识别终端上照一下，相关信息就被读取出来，十分方便。腾讯公司推出的“微信”也支持 QR 码方式生成用户身份信息，还可通过手机扫描 QR 码来添加好友。

习 题 6

1. 简要解释下列名词术语

I/O 设备 并行传输 串行传输 扫描码 逐行扫描法 行列扫描法 像素 (像素)
分辨率 灰度级 字符/数字方式 图形方式 位密度 道容量

2. 简答题

- (1) 磁盘的工作速度是由什么因素决定的?
- (2) 举例说明一种实用的键码形成方法。
- (3) 简述 CRT 显示器由字符代码到字符点阵的转换过程。
- (4) 简述激光打印的基本原理。
- (5) 字符显示器为了实现同步控制, 一般应设置哪几级同步计数器?
- (6) 图形显示器应设置哪几级同步计数器?

3. 某 CRT 显示器作字符显示, 能显示 64 种字符, 每帧可显示最大容量为 25 行 \times 64 字符, 每个字符采用 7 \times 8 点阵, 即横向 7 点, 纵向 8 点, 则字符发生器的容量为多少?

4. 某 CRT 显示器作字符显示, 每帧可显示 25 行 \times 80 列字符, 每个字符采用横 7 \times 纵 9 点阵, 字符间横向间距 2 点, 行间间距 5 点。则点计数器应如何分频? 字符计数器应如何分频? 线计数器应如何分频? 行计数器应如何分频?

5. 某图形显示器的分辨率为横向点 800 \times 纵向点 600, 则同步计数器的点计数器应如何分频? 字节计数器应如何分频? 线计数器应如何分频?

6. 在字符显示器与点阵针打中都有字符发生器, 它们的主要区别是什么?
7. 若要将图形显示的分辨率从 800 \times 600 提高到 1024 \times 1024, 在适配卡上应采取哪些措施?
8. 若要将显示字符从 7 \times 9 点阵放大为 14 \times 18 点阵, 请提出一种实现方案。
9. 若要将显示画面 (字符型) 自下而上地滚动, 请提出一种实现方案。
10. 若要将显示字符 A 从屏幕左上角逐渐地移向屏幕右下角, 请提出一种实现方案。
11. 若要让一个图形在屏幕上旋转, 请提出一种实现方案。

第 7 章 计算机硬件系统模型

前面各章分别讲解了 CPU 子系统、存储子系统、输入/输出子系统和一些常见的外围设备，本章将给出一个简化的模型机系统，并从系统级进行分析：如何通过各大部件之间的互连以构成一个整机系统？通过系统总线如何实现信息传输的方法及其控制机理，从而在系统级上建立起整机概念？本章内容属于总结性质的内容，是对系统一级整机相关内容的集中概括。

7.1 模型机系统及信号互连

为了有利于掌握计算机系统组成的基本原理，我们设计了一台模型机，突出了各主要部件（如 CPU、主存、I/O 设备及接口）之间的连接关系，并大大简化了控制信号，基本上只保留了那些为阐明基本工作原理所必需的信号。

7.1.1 系统组成

图 1-6 给出了常见的计算机硬件系统结构图，在学习了有关知识以后，我们将它具体化为一个模型机，其中各部件之间的逻辑关系如图 7-1 所示。一组被各部件分时共享的系统总线成为连接枢纽，它连接了 CPU、主存、公共接口逻辑、各种接口（适配卡），并通过这些接口连接了几种最常用的外围设备。接口 1 是磁盘适配卡，通过适配卡连接磁盘驱动器。接口 2 是显示器适配卡，连接显示器。接口 3 是打印机适配卡，也可以作为一个通用并行接口，打印机控制器则位于打印机中。接口 4 是一个键盘接口，其接口逻辑很简单，可直接安装在主机板上。接口 5 是一个通信适配卡，其中含有一个以单片机为核心的通信控制器，可编程设置工作方式，实现与本系统之外的设备或网络之间的通信。

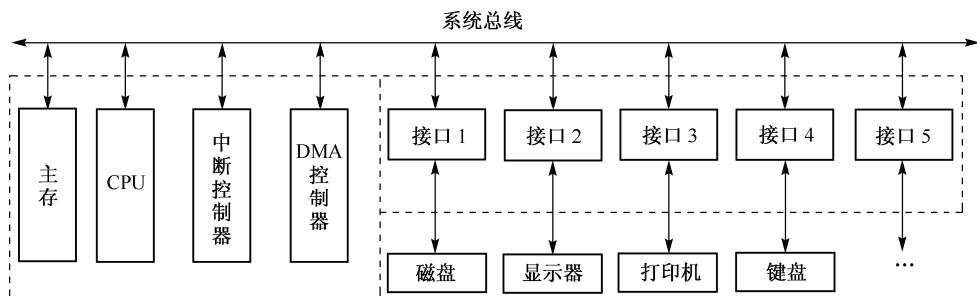


图 7-1 模型机系统框图

从计算机系统组装的角度看，主机的机箱中有一块主机板和几块接口（适配卡）插件。在主机板上安装有 CPU、主存、中断控制器、DMA 控制器等（也可安装键盘接口逻辑），图 7-1 中左虚框内画出了主机板上的组成。右虚框内是几块接口卡，一般将每种接口卡作为一块独立的插件，便于根据需要来对模型机系统进行灵活配置。

随着半导体技术的不断发展，设备的集成度也在不断提高，为了降低成本，一些较为标准化的接口适配卡目前也多被集成到了主机板上，如磁盘接口、显示卡等。磁盘驱动器一般安装在机箱中，键盘、显示器、打印机等则安装在机箱之外。

7.1.2 系统总线

与 5.2.2 节介绍的实际系统总线相比,本章模型机进一步简化,没有采用信号线复用技术,也没有对信号线的设置进行优化,在讨论几种基本工作方式时只选取需要用到的一些信号,并且仅考虑总线的同步控制方式,目的是让读者重点掌握基本原理。

1. 电源线

+5 V, 1 条, 地线 1 条。

2. 地址线

$A_{15} \sim A_0$ 共 16 条, 寻址空间 64 KB (字)。在实际的机器中, 由于字节是信息表示的基本单位, 主存储器通常按字节编址, 本模型机中采用按字编址主要是为了简化系统, 突出基本原理。

3. 数据线

$D_{15} \sim D_0$ 共 16 条, 按字并行传输。

4. 控制信号线

按照其功能性质, 控制信号可分为以下几类。

(1) 复位 $\overline{\text{RESET}}$

复位信号在接通电源时由“电源正常”信号产生, 或由复位键产生。当 $\overline{\text{RESET}}$ 为低电平时, 使系统各部分进入正常的初始状态。

(2) 时序信号

① 时钟 CLOCK: 由 CPU 在每一个时钟周期开始时发出, 作为系统的同步定时信号, 使系统总线按同步控制方式工作。

② 附加同步信号 ASCLOCK: 如果工作需要, 可由外部产生 ASCLOCK, 送入 CPU 以产生系统定时中断。

③ 就绪 $\overline{\text{READY}}$: 由 I/O 接口或主存产生, 当 $\overline{\text{READY}}$ 为低电平时, 通知 CPU 或有关接口已经准备就绪。

(3) 数据传输控制

① 存储器读 $\overline{\text{MEMR}}$ 。

② 存储器写 $\overline{\text{MEMW}}$ 。

③ 输入/输出设备读 $\overline{\text{IOR}}$ 。

④ 输入/输出设备写 $\overline{\text{IOW}}$ 。

(4) 中断控制

① 设备中断请求 8 条 $\text{IREQ}_0 \sim \text{IREQ}_7$: 由各 I/O 接口产生, 送入中断控制器。按图 7-1 的配置, 只需用到 6 条, 分配如下: 系统定时中断请求 IREQ_0 (级别最高)、显示器中断 IREQ_1 、磁盘中断 IREQ_2 、外部通信中断 IREQ_3 、键盘中断 IREQ_4 、打印机中断 IREQ_5 。

② 中断请求 INTR: 由中断控制器产生, 送入 CPU。

③ 中断响应 INTA: 由 CPU 发出, 送往中断控制器。

(5) DMA 控制

① DMA 请求 4 条 $\text{DREQ}_0 \sim \text{DREQ}_3$: 分别由 DMA 接口与主存等产生, 送入 DMA 控制器。由主存的控制逻辑产生动态刷新请求 DREQ_0 (级别最高)、磁盘 DMA 请求 DREQ_1 、外部通信 DMA

请求 DREQ₂, DREQ₃ 暂时保留不用。

② DMA 批准 4 条 DACK₀~DACK₃: 由 DMA 控制器产生, 分别送往提出 DMA 请求的接口, 分配关系同上。

(6) 总线控制

① 总线请求 HRQ: 由 DMA 控制器产生, 送往 CPU。

② 总线批准 HLDA: 由 CPU 发出, 送往 DMA 控制器。

以上共 62 条系统总线。

7.1.3 各部件的信号线

系统总线是公用的, 每个部件根据自身的需要只用到其中的部分信号, 现在我们先分部件进行介绍, 以求加深印象。

1. CPU

系统总线的大部分信号都与 CPU 有关, 当 CPU 设计发生重大变化时, 系统总线的信号组成也随之改变。在实际计算机中, CPU 的出/入信号往往要经过一些缓冲电路和控制信号变换电路再与系统总线连接。本章的模型机则简化为: CPU 与系统总线直接相连。

(1) 输入信号 (由系统总线输入 CPU)

① 电源。

② 复位 $\overline{\text{RESET}}$ 。

③ 附加同步 ASCLOCK。

④ 就绪 $\overline{\text{READY}}$ 。

⑤ 中断请求 INTR。

⑥ 总线请求 HRQ。

(2) 输出信号 (由 CPU 送往系统总线)

① 地址 A₁₅~A₀: 由 CPU 内的 MAR 产生。

② 时钟: CLOCK。

③ 读写命令 $\overline{\text{MEMR}}$ 、 $\overline{\text{MEMW}}$ 、 $\overline{\text{IOR}}$ 、 $\overline{\text{IOW}}$ 。

④ 中断响应 INTR。

⑤ 总线批准 HLDA。

(3) 双向信号

数据 D₁₅~D₀: 可由 CPU 内的 MDR 送往系统总线, 也可由系统总线送入 MDR。

2. 主存储器

许多计算机在 CPU 与主存之间设置了专用的存储总线, 以提高访存速度并减轻系统总线的负担。模型机则简化为: 将主存直接挂载在系统总线上。

(1) 输入信号

① 电源。

② 地址 A₁₅~A₀。

③ 时钟 CLOCK。

④ 存储器读 $\overline{\text{MEMR}}$ 。

⑤ 存储器写 $\overline{\text{MEMW}}$ 。

⑥ 刷新批准 \overline{DACK}_0 。

在第3章中分析CPU指令流程时，我们曾假定CPU时钟周期与主存的读写周期相同，使时序关系得到简化。但现在CPU的工作速度迅速提高，而半导体存储器速度的提高程度远不能满足CPU的需要。因此，本章的模型机采取现在的计算机所普遍使用的方法，让一个主存读写周期占用几个CPU时钟周期。例如，选用读写周期为50 ns的存储芯片，让一个标准的系统总线周期等于一个主存读写周期，并等于8个CPU时钟周期，则可支持百余MHz时钟频率的CPU芯片。

模型机采用动态RAM 64 K×1的芯片17块。

当CLOCK与 \overline{MEMR} （或 \overline{MEMW} ）同时出现时，主存中的一组延时电路依次产生读/写时序。行选信号先将 $A_{15} \sim A_8$ 送入芯片内的行地址锁存器，然后列选信号将 $A_7 \sim A_0$ 送入芯片内的列地址锁存器。由 \overline{MEMR} 与 \overline{MEMW} 组合并延迟发出的读写信号也随后送入芯片，如果既无 \overline{MEMR} 也无 \overline{MEMW} ，则上述读写时序电路将不产生输出，即没有行选与列选信号产生。

(2) 输出信号

① 动态刷新请求 \overline{DREQ}_0 。

② 就绪 \overline{READY} ：如果主存与其他部件之间的数据传输时间无法预先确定（如与系统之外的设备进行通信），则可以通过 \overline{READY} 信号实现应答，以选择所需的时钟周期数。

(3) 双向信号

数据 $D_{15} \sim D_0$ 。

3. 键盘

由键盘输出到主机板上的信号有：键盘数据KD和键盘时钟KC。

键盘矩阵产生的按键位置码送入一个单片机，在其中通过查表转换为按键的ASCII编码，然后通过KD线串行送入主机板上的键盘接口。单片机还输出一个定时信号KC，也送入键盘接口，与主机板的时钟信号相综合，产生定时信号去控制接口中的一个移位寄存器，将串行输入的键码组装成可以并行处理的键码。

4. 中断控制器

中断控制器芯片位于主机板上，但通过系统总线与CPU、各中断源相连接。

(1) 输入信号

① 地址 $A_7 \sim A_0$ ：CPU送来I/O端口地址。

② \overline{IOR} 。

③ \overline{IOW} 。

④ 中断响应INTA。

⑤ 设备中断请求 $\overline{IREQ}_0 \sim \overline{IREQ}_7$ 。

(2) 输出信号

中断请求INTR。

(3) 双向信号

数据 $D_{15} \sim D_0$ 。

当I/O端口地址为00000001时，选择中断控制器中的中断类型码寄存器，可在初始化时经数据线为各中断源分配中断类型码。当I/O端口地址为000000010时，选择中断控制器中的屏蔽码寄存器，经数据线送入屏蔽码。当中断控制器接到设备中断请求时，向CPU提出中断请求INTR，接到CPU发来响应信号INTA后，即当CPU批准请求后，中断控制器通过数据线向CPU送入中

断类型码，作为访问中断向量表的依据。

5. DMA 控制器

DMA 控制器位于主机板上，通过系统总线与 CPU、主存、磁盘接口、通信接口等相连。

(1) 输入信号

- ① 数据 $D_{15} \sim D_0$ 。
- ② 设备 DMA 请求 $DREQ_0 \sim DREQ_3$ 。
- ③ 总线批准 HLDA。
- ④ 就绪 \overline{READY} 。

(2) 输出信号

- ① \overline{MEMR} 和 \overline{MEMW} ：在 DMA 传输阶段，由 DMA 控制器发出，控制主存读/写。
- ② 地址 $A_{15} \sim A_8$ 。
- ③ DMA 批准 $DACK_0 \sim DACK_3$ 。
- ④ 总线请求 HRQ。

(3) 双向信号

① 地址 $A_7 \sim A_0$ ：在初始化阶段，由 $A_7 \sim A_0$ 输入 I/O 端口地址，选择 DMA 控制器中的有关寄存器，经数据线送入有关初始值，如主存缓冲区首址、数据块长度等；在 DMA 传输阶段，由 $A_{15} \sim A_0$ 送出总线地址选取主存单元。

② IOR、IOW：在初始化阶段，由 CPU 送入 DMA 控制器，控制有关寄存器读写；在 DMA 传输阶段，由 DMA 控制器发出，控制有关 I/O 接口的读写。

由于初始化时需送入的预置信息较多，DMA 控制器内的有关寄存器也就比较多，因此我们为模型机 DMA 控制器分配了 16 个 I/O 端口地址：00000011~00010010。

6. 磁盘接口

磁盘接口又称磁盘适配卡，它的一侧与系统总线相连，另一侧与磁盘驱动器相连。

(1) 接口与系统总线之间

- ① 地址 $A_7 \sim A_0$ ：输入，为磁盘接口分配了 4 个 I/O 端口地址：00010011~00010110。
- ② 数据 $D_7 \sim D_0$ ，双向。
- ③ \overline{IOR} 和 \overline{IOW} ，输入。
- ④ $IREQ_2$ ，输出。
- ⑤ $DREQ_1$ ，输出。
- ⑥ $DACK_1$ ，输入。
- ⑦ \overline{READY} ，双向。

在 DMA 初始化阶段，CPU 经系统总线送来 I/O 端口地址和 \overline{IOW} 命令，经数据线 $D_7 \sim D_0$ 送入初始化信息，如磁盘寻址信息、读/写命令、屏蔽信息等。在 DMA 传输阶段，接口输入/输出数据信息。在结束处理阶段或调试阶段，CPU 可通过 \overline{IOR} 命令从接口中读取有关状态信息。

(2) 接口与磁盘驱动器之间

按 SCSI 接口标准，有下列信号：

- ① 数据 $DB_7 \sim DB_0$ ，数据校验位 DBP。
- ② 请求 REQ，对驱动器的调用请求。
- ③ 认可 ACK，驱动器被调用后的回答。

- ④ 应答 ATN, 对发送信息的确认。
- ⑤ 复位驱动器 RST, 使它处于空闲状态。
- ⑥ 选择驱动器 SEL。
- ⑦ 忙 BSY。
- ⑧ C/D, 数据线传输信息是命令或数据。
- ⑨ 输入/输出 I/O, 数据线传输方向。
- ⑩ 信息字节 MSG。

7. 显示器接口 (适配卡)

(1) 接口与系统总线之间

- ① 地址 $A_{15} \sim A_0$, 输入。
- ② 数据 $D_7 \sim D_0$, 双向。
- ③ \overline{IOR} 和 \overline{IOW} , 输入。
- ④ 中断请求 $IREQ_1$, 输出。

显示器适配器中的控制寄存器较多, 模型机为它们分配了 32 个 I/O 端口地址 (实际机器可能更多): 00100000~00111111。VRAM 则占用主存空间高端的一个区域。

在初始化阶段, CPU 通过 $A_7 \sim A_0$ 送来 I/O 端口地址, 选择适配器中的某个控制器; 通过 $D_7 \sim D_0$ 送入预置信息, 以设置适配卡的工作方式、显示规格等, 相应地确定了各同步计数器的分频关系。在显示期间, 利用回扫消隐期以中断方式送入 VRAM 的地址与显示信息, 对 VRAM 进行内容刷新。

(2) 由适配器送往 CRT 显示头

视频信号、亮度控制、水平同步、垂直同步等。

8. 打印机接口

(1) 接口与系统总线之间

- ① 地址 $A_7 \sim A_0$, 输入接口。
- ② 数据 $D_7 \sim D_0$, 双向。
- ③ \overline{IOR} 和 \overline{IOW} , 输入。
- ④ 中断请求 $IREQ_5$, 输出。

为打印机接口分配 5 个 I/O 端口地址: 00011000~00011100。进行打印机初始化时, CPU 送出 I/O 端口地址, 通过 $D_7 \sim D_0$, 送入命令字。完成打印机初始化后, 通过 $D_7 \sim D_0$ 送入打印信息如字符编码。

(2) 接口与打印机之间

- ① 数据 $DP_7 \sim DP_0$, 接口输出。
- ② 命令 $CP_7 \sim CP_0$, 接口输出。
- ③ 状态 $SP_7 \sim SP_0$, 输入接口。

9. 通信适配卡

(1) 适配卡与系统总线之间

- ① 地址 $A_7 \sim A_0$, 输入。
- ② 数据 $D_7 \sim D_0$, 双向。

- ③ $\overline{\text{IOR}}$ 和 $\overline{\text{IOW}}$ ，输入。
- ④ 中断请求 IREQ_3 ，输出。
- ⑤ DMA 请求 DREQ_2 ，输出。
- ⑥ DMA 批准 DACK_2 ，输入。

(2) 适配卡与通信线路之间

数据线 $\text{D}_7 \sim \text{D}_0$ ，双向。

通信适配器的实际设计要考虑许多因素，如通信规范、控制方法等。本模型机只对此作示意性设计。

本模型机的设计主要立足于让读者对计算机系统的组成有一个比较完整的、清晰的整机概念，因而比实际机器做了许多简化。虽然它以当前的主流微机技术为背景，但所做的信号约定和端口地址分配等不一定与实际机器的细节一致。

7.2 模型机典型 I/O 操作举例

第 5 章中已经阐明了 3 种 I/O 传输控制机制，现在以模型机为具体对象分析 3 种控制机制的典型操作，以求从工作机制角度建立系统级的整机概念。

7.2.1 直接程序传输方式的 I/O 操作

直接程序传输方式是指在程序中依靠 I/O 指令（显式或隐式 I/O 指令）实现主机与外围设备之间的调用和对数据输入/输出操作的控制。

模型机对外围设备的调用过程：在输出指令中给出某个 I/O 端口地址，如 $\text{A}_7 \sim \text{A}_0 = 00011000$ 。由于该端口地址是分配给打印机接口的，所以经译码选中了接口中的命令字寄存器。输出指令操作码产生 $\overline{\text{IOW}}$ 命令，指令所指定的 CPU 寄存器中已经准备好命令字代码，在 $\overline{\text{IOW}}$ 作用下命令字经 $\text{D}_7 \sim \text{D}_0$ 送入接口的命令字寄存器。其中启动位为 1，于是接口产生启动命令送入打印机，让打印机执行一个初始化程序，实现对打印机的调用。当初始化完成后，打印机处于可接收代码状态，CPU 执行输出指令，其中给出另一个 I/O 端口地址，如 00011001 ，将一个打印字符编码送入接口中的数据寄存器，再由接口送入打印机。接口状态寄存器记下数据寄存器已空，可以再次接收打印字符编码。

怎样知道打印机已经处于可接收代码状态呢？在等待这一状态出现之前 CPU 又干什么？在直接程序控制方式中，CPU 采取“查询—等待”策略，即 CPU 在送出调用打印机的命令字后，就执行一条输入指令从接口调回状态字以供判别。例如，由输入指令产生 $\overline{\text{IOR}}$ 命令和 I/O 端口地址 00011010 ，选中接口的状态寄存器，将状态字调入 CPU 某一寄存器中，然后执行程序对状态字进行判别。若打印机接口还不能接收打印字符编码，则 CPU 重复执行读取接口状态字、分析判断的程序段，这就是“查询—等待”。如果打印机已经完成初始化或是已从接口中取走一个字符编码，则状态字发生相应变化，CPU 判明之后转入输出信息程序段，从主存的打印缓冲区中读取一个字符编码到 CPU 中，再执行一条输出指令，向接口送出打印字符编码。然后又重复上述操作。

系统总线工作周期有 3 种可供选择的类型。

- ⊙ 短周期：与 CPU 时钟周期相同，用于 CPU 与接口之间的数据传输。上例采用短周期。
- ⊙ 标准周期：与主存读写周期相同，用于 CPU 访存或主存与 I/O 接口之间的数据传输。
- ⊙ 扩展周期：所占时钟周期数视需要而定，由就绪信号 $\overline{\text{READY}}$ 控制其长短。

7.2.2 中断方式下的 I/O 操作

用于 I/O 的中断方式是通过执行中断处理程序来控制 I/O 操作的，其中包括对有关工作状态的分析判断，做出相应决策，通过 I/O 指令实现 I/O 传输。除了这些必要的与 I/O 相关的操作外，CPU 在其他时间里可以继续执行原来的程序，从而在一定程度上可以让 CPU 与外围设备并行工作。现在仍以打印机工作为例进行说明。

1. 调用打印机

CPU 执行输出指令，将准备好的命令字送往打印机接口的命令字寄存器，接口再将有关命令送往打印机，打印机开始执行初始化程序。这一过程与直接程序控制方式的调用过程完全相同。

在打印机执行自己的初始化程序期间，CPU 不必光等待不做事，它可以继续执行自己的程序进行运算处理。因为一旦打印机完成初始化，会以中断请求信号通知 CPU。

2. 中断请求、响应、处理

当打印机完成初始化后，相应的状态字发生变化，其中的“忙”位 $\overline{\text{BUSY}}$ 变为高电平，表示不忙。接口根据这一状态产生中断请求 IREQ_5 ，送往主机板上的中断控制器。中断控制器发出 INTR ，送入 CPU。这一中断请求何时产生，CPU 事先并不知道，所以我们说：中断是随机产生的，打印机所要求的中断服务是以随机事件的形式出现的。

假定 CPU 在执行一条指令的过程中接到了中断请求信号 INTR ，而且具备响应中断的条件，则在这条指令即将执行完毕的一个时钟周期里向中断控制器发出批准信号 INTA ，并在指令结束后转入一个过渡周期即中断周期 IT 。在 IT 中 CPU 先关中断，然后将程序断点（当前 PC 的内容）压入堆栈；通过数据线 $\text{D}_7 \sim \text{D}_0$ 从中断控制器取回打印机中断的中断类型码（这一类型码是在系统初始化时预置的），类型码乘以 2 得到向量地址 $2n$ （每个中断向量占两个字，分别是中断处理程序的入口地址和程序状态字），将这一地址送入 MAR，据此访问中断向量表（位于主存储器中），从中取出打印中断处理程序入口地址，送入程序计数器 PC。然后转入新的取指周期 FT ，开始执行打印中断处理程序。

在打印中断处理程序中，CPU 用输入指令调回打印机接口中的状态字，分析打印机现在所处的状态，从主存中的打印缓冲区读出一个字符编码，用输出指令送往打印机接口。接口再将该字符编码送入打印机，打印机回答一个确认信号 ACK ，接口中状态寄存器相应位 ACK 为 1。据此，CPU 继续输出打印字符编码，直到一行打印信息送完为止。

3. 打印机打印，CPU 并行工作

打印机开始执行打印程序，CPU 从堆栈中取出返回地址，返回并继续执行主程序。

4. 再次中断

当打印完一行字符后，接口状态寄存器中的 $\overline{\text{BUSY}}$ 再度变为高电平，接口再次提出中断请求 IREQ_5 。

按照上述方式执行，直到打印缓冲区中的信息全部打印完为止。

以上描述主要着重于 I/O 部分，对一些细节如缓冲区设置、中断控制器预置、处理程序中的一些分析判别、打印机自身的控制程序等，做了许多简化。

在中断方式中，当打印机执行初始化程序以及打印机正在打印时，可以让 CPU 和打印机并行工作。当 CPU 接到中断请求后，需对打印机有关状态进行判别，并向打印机输出打印信息时，

CPU 必须暂停原程序转去执行中断处理程序。

7.2.3 DMA 方式下的 I/O 操作

DMA 方式是直接依靠硬件实现主存与某些 I/O 设备之间的数据直传,常见的方式是将系统总线交由 DMA 控制器进行控制,让 I/O 接口与主存直接通过系统总线进行数据传输,不经过 CPU。为此,需事先通过程序预置有关信息,即 DMA 初始化,并在批量传输结束时通过程序进行结束处理。

我们以模型机的磁盘调用为例说明 DMA 方式下的 I/O 操作。

1. DMA 初始化

CPU 通过输出指令预置 DMA 控制器本身的工作方式。指令给出的 I/O 端口地址选择 DMA 控制器有关寄存器,写入工作方式代码(细节略)。

CPU 通过输出指令向 DMA 控制器预置:传输方向(是 $M \rightarrow$ 接口,还是接口 $\rightarrow M$)、主存缓冲区首址、批量传输字节数。

CPU 通过输出指令向磁盘接口预置:磁盘寻址信息(驱动器号、磁头号、起始扇区号等),读/写命令输出指令给出 I/O 端口地址选择接口寄存器。

2. 启动磁盘工作

CPU 通过输出指令向磁盘接口送出命令字,其中启动位为 1。接口据此向驱动器发出启动命令,开始寻道。此后,CPU 可以继续执行原来的程序进行运算处理。

3. 磁盘读/写, CPU 并行工作

磁盘寻道,当寻道完成并判明无误后,等待起始扇区经过磁头。找到起始扇区后,开始连续地读/写。

读盘:磁盘将读出的数据送入接口中的一个缓冲存储器。缓存容量不小于 1 KB,即两个扇区容量。“装满一个扇区容量”是读盘时接口提出 DMA 请求 $DREQ_1$ 的条件。

写盘:“缓存有一个扇区空”是写盘时接口提出 DMA 请求 $DREQ_1$ 的条件。因此一开始就提出请求,等到缓存中存有写入数据之后才开始将数据送入驱动器。

在磁盘读/写过程中,CPU 与磁盘并行工作。

4. DMA 请求、响应、批量传输

$DREQ_1$ 送往 DMA 控制器,然后控制器提出总线请求 HRQ 送往 CPU。若具备响应条件,则 CPU 在一个系统总线周期结束时予以响应,发出总线批准信号 $HLDA$,送回到 DMA 控制器。同时,CPU 的 MAR 输出端呈现高阻抗,与系统总线脱钩,将总线控制权交给 DMA 控制器。

DMA 控制器接管总线控制权后,一方面向磁盘接口送出批准信号 $DACK_1$,另一方面送出地址信息到地址总线,并发出读写命令。如果是读盘,则控制器发出 \overline{MEMW} 和 \overline{IOR} ,即读出磁盘接口缓存数据,写入主存缓冲区。如果是写盘,则发出 \overline{MEMR} 和 \overline{IOW} 。

第一次给出的地址是主存缓冲区首址,每完成一次传输,DMA 控制器内的地址寄存器内容加 1,传输字节数减 1。当传输字节数为 0 时,表明这次传输已经结束,DMA 控制器终止传输,将总线控制权交还给 CPU。

在 DMA 传输过程中 CPU 干什么?如果模型机 CPU 没有预存指令与数据的功能(如第 3 章 CPU 模型那样的档次),则 CPU 进入一连串的 DMA 周期,暂停工作,但不需保存断点与现场。

如果 CPU 内有 Cache, 存有当前需执行的程序和数据, 虽然系统总线已经交给了 DMA 控制器, CPU 仍可并行工作。

5. 以中断方式进行结束处理

当 DMA 批量传输结束后, 磁盘接口提出中断请求 IREQ₂, 中断控制器据此提出 INTR, CPU 响应, 执行磁盘中断处理程序。该处理程序将调回磁盘接口状态信息, 判断刚才的 DMA 传输有无错误或故障。若状态信息表明正确无误, 则此次磁盘调用成功。若状态信息表明有错误, 则转入出错处理, 例如重新让磁盘定位, 再读/写一遍等。

至此, 我们分别描述了三种典型工作机制下的 I/O 过程。

7.3 系统配置举例

在第 1 章的开始就提到, 大家可能已经在某种程度上使用过计算机, 当时只是从外观上了解计算机。随后本书各章分析了计算机的内部组成和工作原理。现在提出一个问题, 如果你需要购置一台微型计算机, 应当如何选择系统的配置? 显然, 这要根据自己的实际需要以及打算付出的费用来决定。下面以本书编写时的市场情况为前提, 通过一种实际系统的配置, 来介绍购置机器时应该考虑的因素, 以此作为全书的结束。

(1) 机箱 (带电源), ATX 规格

机箱的规格决定了它能容纳什么样的主板以及磁盘等设备, 当前流行 ATX 规格, 意为支持 ATX 规格主板。

(2) 主板

购置机器时, 主板可以说是最重要的部件, 选购主板需要了解: 主板采用的芯片组、支持的 CPU 类型、支持的内存标准、支持的接口等; 随着集成度的不断提高, 目前主板上集成了越来越多的功能, 如显卡、声卡、网卡等。综合考虑, 选择 MSI (微星) B85M-E45 主板, 标准 LGA 1150 接口, 兼容 Core i3 处理器, 内置 4×DDR3 内存插槽。

① CPU: Core i3-3240T, 主频 2.9 GHz, 双核, 片内三级 Cache 3 MB。

在选择 CPU 时, 应注意哪些产品已属于淘汰型, 哪些属于快被淘汰的“鸡肋”型, 哪些虽已出现但尚未成熟。截止到 2013 年年底, Intel 公司的 Pentium 系列微处理器已属淘汰型, Core 2 DUO 双核系列产品已处于“鸡肋”之列。目前, Core i7/i5/i3 双核、四核系列 CPU 产品正处于主流地位; AMD 的“速龙”和 APU 系列产品则具有极高的性价比, 非常适合低端用户及家庭用户, 对 Intel 形成了一定的威胁。目前, CPU 以 64 位双核和四核为主, Intel 和 AMD 等公司都已推出了自己的 64 位处理器产品。由于价格及应用软件等方面的原因, 64 位的八核以上 CPU 还主要应用在一些高档服务器上。

② 内存: 4 GB, DDR 3, 频率 1600 MHz。

4GB 的容量能支持运行规模较大的系统软件, 如 Windows 8 和一些三维图形应用软件。目前, DDR2 已逐渐被淘汰, 采用 DDR 技术的市场商品主要有 DDR3 1333 MHz、DDR3 1600 MHz 和 DDR3 1866 MHz 等技术规格。

③ 接口: 配置 1 个并行接口, 1 个串行接口, 2 个 USB 2.0 接口, 2 个 SATA 接口, 以及 1 个 IEEE 1394 接口。

USB (Universal Serial Bus, 通用串行总线) 接口并不是一种新的总线标准, 而是计算机系统连接外围设备 (如键盘、鼠标、打印机等) 的输入/输出接口标准。由于 USB 接口具有外设安装

简单、支持热插拔、速度快（USB 2.0 理论上可高达 480 Mb/s，一般能达到 100 Mb/s）、连接距离达 5 m、支持多设备连接、内置电源等诸多特性，因而有逐步取代普通串行口和并行口的趋势。IEEE 1394 也是一种高效的串行接口标准，具有许多类似于 USB 接口的特性和优点，其速度则更快，并能连接成为一个小型的网络。因此，将来很可能会用 IEEE 1394 来连接高速装置及智能家电设备，而 USB 用来连接低速装置，以此配合来达到较佳的性价比。一个 IDE 接口可以主从方式连接两台设备，如磁盘、光盘。目前，在个人计算机中广泛使用 IDE 接口标准，在服务器中则多使用 SCSI 接口。

（3）硬盘驱动器

配置 Seagate（希捷）Desktop HDD 系列，容量 1 TB，7200 r/min，64 MB 缓存，SATA 接口标准，速度 6 Gb/s。

（4）外设存储器

目前，一般的用户都会配置 U 盘和移动硬盘。

U 盘：朗科朗科 U903，8 GB 容量，USB 3.0 接口。

移动硬盘：配置 Seagate（希捷）睿品系列，2.5 英寸，容量 1 TB，USB 3.0 接口。

（5）光驱：48X CD-RW/DVD/CD-ROM

目前，支持 DVD-ROM 的光驱已逐渐成为市场的主流，许多产品不但兼容了 CD-ROM，更集成了 CD-RW 刻录功能。

（6）显卡

配置 GIGABYTE（技嘉）NVIDIA GeForce GTX 750，核心频率 1033 MHz，显存容量 2 GB，显存类型 GDDR5，显存频率 5400 MHz，显存位宽 128 bit，分别带 DVI 接口 2 个和 HDMI 接口 2 个，支持的最大分辨率 2560×1600。

目前，显卡内的核心芯片其市场份额主要被 NVIDIA 和 AMD（前生为 ATI）两家公司所占有。其中，NVIDIA 公司的采用 Geforce TITAN 架构的系列显卡显示控制新能极好，其核心频率高达 967 MHz，有 2880 个流处理单元，显存采用 GDDR5 类型，容量 6144 MB、位宽 384 位、频率高达 7000 MHz，且支持 DirectX 11.2 API 和 OpenGL 4.4。但这类高端显卡价格很贵，一般只在高端图形工作站上使用。NVIDIA 在收购了以 Voodoo 系列显卡闻名于世的 3DFX 公司后，更是所向无敌，但近来 ATI（后被 AMD）公司的 R9 系列显卡对 NVIDIA 发起了强有力的挑战，为用户提供了更多的选择。

（7）彩色显示器

配置 SAMSUNG（三星）S22B310B 液晶显示器，21.5 英寸，16:9 宽屏，其最佳分辨率为 1920×1280，0.248 mm 点距，颜色数 16.7M，亮度 250 cd/m²，对比度为 1000000:1，其可视角度范围 170°/160°（CR>10 时）。目前，液晶显示器以其无闪烁、纤薄美观等优点，获得了包括家庭用户在内的许多用户的青睐，也已成为市场上的主流产品。

（8）声卡

配置华硕 Xonar D2 声卡，24 位，支持杜比、DTS 及 3D 音效，Hyper-Grounding 布线设计，具备清晰的在线游戏和语音聊天效果。

（9）键盘

配置罗技（Logitech）K120 有线键盘，USB 接口。

（10）鼠标

配置 Logitech（罗技）M100 型光电有线鼠标，鼠标分辨率 1000 dpi，USB 接口。

(11) 扫描仪

配置紫光 LA2000 扫描仪, 4800×1200 dpi 光学分辨率, A4 扫描幅面, CMOS 光源, 颜色数 48 位真彩色, USB 2.0 接口, 且配置 OCR (印刷体字符自动识别) 软件。

(12) 打印机

彩色喷墨式, 4800 dpi 点距, 10 页/分, 支持 USB 接口。

以目前技术而言, 除了需要票据套打功能以外, 针式打印机已逐渐被淘汰, 喷墨式和激光打印式已成主流。就打印机本身的价格而言喷墨式较有吸引力, 而激光打印机则具有打印黑白文档质量高, 平均打印成本低的优势。

(13) 网卡 (NIC): 10 M/100 M 自适应 PCI 网卡

10 M/100 M 自适应 PCI 网卡能自动识别网络速度, 不需人为设定就可以自动工作在 10 Mbps 或 100 Mbps 带宽中。目前, 无线网卡的价格下降很快, 普通用户完全可以承受, 因此逐渐成为需要经常移动办公的人士的理想选择, 甚至可用来构建灵活的家庭无线局域网, 但在信息安全等方面还有待改善。

(14) 基本软件

① 操作系统: Microsoft Windows 8。

② 办公软件: Microsfot Office 2013 应用套件, 其中包含字处理软件 Word 2013、电子表格软件 Excel 2013 和电子邮件软件 Outlook 2013 等。

③ 网络浏览器: Google Chrome 2013。

④ 其他常用软件: 包括多媒体播放软件、各种上网工具等。

以上配置达到了通常所说的多媒体计算机标准。

注意, 计算机技术的发展极其迅速, 新的产品层出不穷, 性能指标日新月异, 具体选购时必须以当时的情况为依据。

习 题 7

1. 以打印机为例, 阐述中断方式工作的全过程。
2. 以读盘为例, 阐述 DMA 方式工作的全过程。
3. 试用流程图描述键盘按中断方式工作的全过程。
4. 试用流程图描述写盘按 DMA 方式工作的全过程。
5. 查阅有关资料, 进一步解释 7.3 节实例中的有关技术指标与接口标准的含义。

参考文献

- [1] 王诚. 计算机组成与设计. 北京: 高等教育出版社, 2011.
- [2] 蒋本珊. 计算机组成原理 (第3版). 北京: 清华大学出版社, 2013.
- [3] 唐朔飞. 计算机组成原理 (第2版). 北京: 高等教育出版社, 2008.
- [4] 白中英, 戴志涛. 计算机组成原理 (第五版). 北京: 科学出版社, 2013.
- [5] 王爱英. 计算机组成与结构 (第5版). 北京: 清华大学出版社, 2013.
- [6] David A. Patterson, John L. Hennessy. 计算机组成与设计 (第4版). 康继昌, 樊晓桢, 安建峰等译. 北京: 机械工业出版社, 2011.
- [7] 马维华等. 从 8086 到 Pentium III 微型计算机及接口技术. 北京: 科学出版社, 2002.
- [8] 周明德. 微型计算机系统原理及应用 (第五版). 北京: 清华大学出版社, 2007.
- [9] 李继灿. 新编 16/32 位微型计算机原理及应用. 北京: 清华大学出版社, 2008.
- [10] 百度百科: <http://baike.baidu.com>
- [11] 维基百科: (中) <http://zh.wikipedia.org/wiki/Wikipedia>; (英) http://en.wikipedia.org/wiki/Main_Page

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，本社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：(010) 88254396; (010) 88258888

传 真：(010) 88254397

E-mail: dbqq@phei.com.cn

通信地址：北京市海淀区万寿路 173 信箱

电子工业出版社总编办公室

邮 编：100036



欢迎登录 免费获取本书教学资源
<http://www.hxedu.com.cn>

计算机组成原理 (第3版)

Computer composition and principle

本书为普通高等教育“十一五”国家级规划教材。

本书以当前主流微型计算机技术为背景，以建立系统级的整机概念为目的，深入介绍了计算机各功能子系统的逻辑组成和工作机制。全书共分7章。第1章概述计算机的基本概念和计算机系统的硬件、软件组织；第2章介绍数据信息和指令信息的表示方法；第3章介绍CPU子系统；第4章介绍存储子系统，讨论存储原理和主存储器的设计方法；第5章介绍I/O子系统，包括接口的基本知识，系统总线，以及中断和DMA等I/O传送控制机制；第6章介绍常用输入/输出设备的工作原理及信息转换过程；第7章以一个计算机硬件系统模型作为全书的总结。

本书可作为高等院校计算机及相关专业“计算机组成原理”及相关课程的教材，也可作为从事计算机专业的工程技术人员的参考书。



策划编辑：章海涛
责任编辑：章海涛
责任美编：李 雯

ISBN 978-7-121-23471-2



9 787121 234712 >

定价：45.00 元